DEFENCE RESEARCH ESTABLISHMENT
CENTRE DE RECHERCHES POUR LA DÉFENSE
VALCARTIER, QUÉBEC

DREV TM 1999-125
Unlimited Distribution / Distribution illimitée

GIS PROTOTYPING WITH GLOBAL GEOMATICS (L.A.S.) PRODUCTS

by

G. Thibault and M. Lizotte

December / décembre 1999

Approved by / approuvé par

Section Head / Chef de section

7 12 99
Date

RESTRICTIVE LEGEND

GOVERNMENT OF CANADA

GOUVERNEMENT DU CANADA

THIS DOCUMENT AND THE INFORMATION IT CONTAINS IS DISCLOSED IN CONFIDENCE TO:

FOR DEFENCE INFORMATION PURPOSES ONLY, WITH NO DISCLOSURES TO OTHERS OR ANY OTHER USE (INCLUDING ANY MANUFACTURING USE) TO BE MADE WITHOUT THE WRITTEN PERMISSION OF THE CANADIAN DEPARTMENT OF NATIONAL DEFENCE (via DRDIM 3).
THIS DOCUMENT CONTAINS PROPRIETARY INFORMATION.

WARNING NOTICE

The information contained herein is proprietary to Her Majesty and is provided to the recipient on the understanding that it will be used for information and evaluation purposes only. Any commercial use, including use for manufacture, is prohibited. Release to third parties of this publication or of information contained herein is prohibited without the prior consent of DND Canada.

© Her Majesty the Queen in Right of Canada as represented by the Minister of National Defence, 1999

ABSTRACT

Any automated Command and Control Information System (C2IS), in order to be successful, must rely on an efficient geographical information system (GIS). The purpose of this Technical Memorandum (TM) is to present the results of prototyping activities aimed at exploring one technological alternative with the Global Geomatics (L.A.S.) family of products. It presents Raster/Vector display alternatives and military symbology alternative solutions. Many C++ scripted solutions, including the implementation rules, and the performance evaluations are provided. This study covers the R&D done during the Jan-Apr 1997 period.

RÉSUMÉ

Tout système automatisé de commandement et contrôle doit pouvoir s'appuyer sur un système de géomatique efficace pour avoir du succès. Le but de ce mémo technique est de présenter les résultats des activités de prototypage explorant une alternative technologique avec la famille d'outils de géomatique de Global Geomatics (L.A.S.). Le rapport documente plusieurs solutions alternatives pour la présentation des symboles militaires sur les cartes de format matriciel et vecteur. Plusieurs de ces solutions y sont explicitement données en langage de programmation C++, incluant les règles d'implémentation et les résultats des performances observées lors des tests d'évaluation. Cette étude couvre des activités de R&D faites durant la période de janvier à avril 1997.

TABLE OF CONTENTS

ABSTRACT/RÉSUMÉ i

EXECUTIVE SUMMARY v

1.0 INTRODUCTION 1

1.1 Overview 1
1.2 Conventions 3
1.3 Issues Identified in the Previous Studies 3

2.0 RASTER/VECTOR DISPLAY ALTERNATIVES 6

2.1 Goal 6
2.2 Result Summary 6
2.3 Approach Used 7
2.4 Recommended Approach 11
2.5 Implementation Rules (Annex B) 11

3.0 IMPORT OF ADRG FILES FROM GEOMATICS CANADA CD ROM 12

3.1Goal 12
3.2 Results Summary 12
3.3 Approach Used 12
3.4 Recommended Approach 14
3.5 Implemented Rules (Annex B) 15

4.0 GIS TOOLBOX BUILDING 16

4.1Goal 16
4.2 Results Summary 16
4.3 Approach Used 16
4.4 Recommended Approach 27
4.5 Implemented Rules (Annex B) 27

5.0 SYMBOLOGY ALTERNATIVE SOLUTIONS 28

5.1Goal 28
5.2 Results Summary 28
5.3 Approach Used 28
5.4 Recommended Approach 37
5.5 Implemented Rules 37

6.0 SIMULATE THE CLASS ENCAPSULATION OF RDBMS 38

6.1Goal 38
6.2 Results Summary 38
6.3 Approach Used 38
6.4 Recommended Approach 40
6.5 Implemented Rules 40

7.0 INITIAL DISPLAY OF TACTICAL DATA 41

7.1Goal 41
7.2 Results Summary 41
7.3 Approach Used 41
7.4 Recommended Approach 43
7.5 Implemented Rules (Annex B) 43

8.0 TOOLBOX - SELECT AND QUERY FUNCTIONALITY 44

8.1Goal 44
8.2 Results Summary 44
8.3 Approach Used 44
8.4 Recommended Approach 46
8.5 Implementation Rules (Annex B) 46

9.0 TOOLBOX - PLACE UNIT SYMBOL FUNCTIONALITY 47

9.1Goal 47
9.2 Results Summary 47
9.3 Approach Used 47
9.4 Recommended Approach 48
9.5 Implementation Rules (Annex B) 49

10.0 SITUATION MAP REFRESHES 50

10.1Goal 50
10.2 Results Summary 50
10.3 Approach Used 50
10.4 Recommended Approach 55
10.5 Implementation Rules (Annex B) 55

11.0 COORDINATE MANAGEMENT (MAP PROJECTION, ETC.) 56

11.1Goal 56
11.2 Results Summary 56
11.3 Approach Used 56
11.4 Recommended Approach 57
11.5 Implementation Rules 57

12.0 AREA SYMBOLS DISPLAY AND PLACEMENT 57

13.0 CONCLUSION.....58

14.0 REFERENCES 61

ANNEX A - THE INTERNAL STRUCTURE A-1

ANNEX B - IMPLEMENTATION RULES.....B-1

ANNEX C - PERFORMANCE & STATISTICS.....C-1

FIGURES AND TABLES

EXECUTIVE SUMMARY

Intelligence is an element of operations and an important component of decision-making. It enables commanders to successfully plan and conduct operations, identify targets, and win decisive battles. Ten years ago, the Defence Research Establishment Valcartier (DREV) began a research project to explore means to provide assistance to intelligence analysts responsible for the fusion of tactical information. This investigation led to the development of an experimental processing environment called the "Tactical Information Fusion" (TIF) prototype [1].

The purpose of the TIF prototype was to experiment with and provide tools to the All Source Analysis Team. The tools were experimented in order to improve decision-making, to accelerate registration and analysis of enemy activities, and to assist the Senior Analyst (SA) and the Duty Officer (DO) in the production of intelligence. There were two major field trials conducted in 1994 and 1995, plus some minor evaluation exercises that were conducted during the 1994-1997 period with the TIF prototype. The scope of the trials was voluntarily limited to the evaluation of the SA and the analyst functions, thus concentrating mostly on the processing phase of the Intelligence Cycle [2-4].

Other R&D activities were initiated to continue the enrichment of the corporate knowledge. DREV has designed and produced automated tools to support the development of a modern tactical intelligence processing and analysis capability for the Canadian Land Force. Under the DREV "Advanced Concepts for Tactical Information Fusion" (ACTIF) project many studies were conducted to assist in the formulation of a strategic plan for a tactical intelligence system [5] and a technological infrastructure architecture was developed [6] to support this plan. The design for a future automated tactical

intelligence system is underway and is well documented [7-9]. The concepts, definitions and requirements for a future tactical intelligence system were investigated in a previous DREV study [10].

If there is one lesson to be learned from all the experience DREV has gained during those years conducting R&D in the command and control arena is the following one. In the future, for any automated Command and Control Information System (C2IS) to be successful, it will have to rely on an efficient geographical information system (GIS). Therefore, three studies were conducted to identify promising GIS technological alternatives for the ACTIF architecture [11-13]. The purpose of this Technical Memorandum (TM) is to present the results of prototyping activities aimed at exploring one technological alternative with the "Logiciels et Applications Scientifiques" (L.A.S.) family of products. The L.A.S. company no longer exists and has been bought over by Global Geomatics. It presents raster and vector display alternatives for military symbology. Many C++ scripted solutions, including the implementation rules, and the performance evaluations are provided. This study covers the R&D done during the Jan-Apr 1997 period.

1.0 INTRODUCTION

1.1 Overview

Ten years ago, the Defence Research Establishment Valcartier (DREV) began a research project to explore means to provide assistance to intelligence analysts responsible for the fusion of tactical information. This investigation led to the development of an experimental processing environment called the "Tactical Information Fusion" (TIF) prototype [1].

The purpose of the TIF prototype was to experiment with and provide tools to the All Source Analysis Team. The tools were experimented in order to improve decision-making, to accelerate registration and analysis of enemy activities, and to assist the Senior Analyst (SA) and the Duty Officer (DO) inn the production of intelligence. There were two major field trials conducted in 1994 and 1995, plus some minor evaluation exercises that were conducted during the 1994-1997 period with the TIF prototype. The scope of the trials was voluntarily limited to the evaluation of the SA and the analyst functions, thus concentrating mostly on the processing phase of the Intelligence Cycle [2-4].

Other R&D activities were initiated to continue the enrichment of the corporate knowledge. DREV has designed and produced automated tools to support the development of a modern tactical intelligence processing and analysis capability for the Canadian Land Force. Under the DREV "Advanced Concepts for Tactical Information Fusion" (ACTIF) project many studies were conducted to assist in the formulation of a strategic plan for a tactical intelligence system [5] and a technological infrastructure architecture was developed [6] to support this plan. The design for a future automated tactical intelligence system is underway and is documented [7-9]. The concepts,

definitions and requirements for a future tactical intelligence system are given in a previous DREV study [10].

Any automated Command and Control Information System (C2IS), in order to be successful, must rely on an efficient geographical information system (GIS). Therefore, three studies were conducted to identify promising GIS technological alternatives for the ACTIF architecture [11-13]. The purpose of this Technical Memorandum (TM) is to present the results of prototyping activities aimed at exploring one technological alternative with the "Logiciels et Applications Scientifiques" (L.A.S. now part of Global Geomatics) family of products.

The exploration was conducted following eleven (11) topics. These topics answer issues raised in two previous ACTIF GIS Studies [11-12] and other important issues related to the ACTIF architecture.

All results are documented in this report and some were also integrated into a common environment called the «Situation Map Prototype». This documentation has mainly two objectives:

a) Determine how well Grassland meets the ACTIF architecture needs; and

b) Document the know-how acquired during this prototyping activity in order to benefit from this learning if Grassland[1] is finally selected for ACTIF.

[1] Grassland is a registered trade mark of Logiciels et Applications Scientifiques (L.A.S.) Co now under Global Geomatics.

The functional aspect was not really addressed by the prototyping team. When possible, the ACTIF functional approach was adopted but otherwise, no time was taken to validate the functional approach. For that reason, rules that will be documented are technological i.e. related to the technical integration of the Grassland products.

The first section summarizes the results concerning the issues identified in the first GIS study [11]. Each remaining section of the document is divided into the following subsections:

.1 Goal;

.2 Results Summary;

.3 Approach Used;

.4 Recommended Approach;

.5 Implementation Rules (detailed in Annex B – Implementation Rules).

1.2 Conventions

This document follows specific typographic conventions, outlined below:

Bold is used for directories, headings, highlighting mandatory information in Makefile;

Italic is used to highlight source code examples, variable parameters;

" is used to show Grassland Application program Interface (API) functions; and

□ Text within frame indicates how issues were addressed within the current project.

1.3 Issues identified in the previous studies

There were three possible risk factors identified in the second GIS study [12], namely:

a) The small size and uncertain future of the company in the GIS marketplace;

b) Uncertainty concerning display speed for large raster files; and

c) Uncertainty as to achievable levels of product integration possible on a UNIX platform.

1.3.1 Position in GIS marketplace

Because of its small size and relative youth, "Logiciels et Applications Scientifiques" is not in the first rank of GIS providers. Their business plan appears to be built around a single GIS product: Grassland. The Grassland product cannot be considered mature. Release two was announced in September 1996. The degree of acceptance of this product in the marketplace is not yet clear. Hence, there is some uncertainty regarding the long-term viability of the company and the product.

The basis of Grassland is GRASS, which is a mature and well-known public-domain GIS product. Initial and ongoing development of GRASS was supported until recently by the U.S. Army Corps of Engineers. That direct support is being discontinued. The Corps of Engineers has established a Cooperative Research and Development Agreement (CRADA) with L.A.S. for the maintenance and evolution of GRASS. External funding provided to L.A.S. under the Agreement will help ensure product evolution.

However, in the event that this funding was discontinued, L.A.S. would not necessarily have the resources to further evolve the GIS functionality so as to remain competitive in the marketplace against larger GIS providers.

1.3.2 Questions on raster display speed.

Grassland seems to offer acceptable display refresh speed for raster data but this has been demonstrated only on small files corresponding to small

land areas (smaller than 1Mb). It also seems to lack the richness of functionality that is required to properly manipulate the tiling scheme of the geographic library. This functionality could be important if the existing display speed were not fast enough to meet ACTIF standards. If this product was to be chosen, testing would be required early in the development process in order to find the optimum tiling setup for raster files.

During the current project, a prototype was developed using ADRG raster files. L.A.S. offered an Open Geospatial Datastore Interface (OGDI) which could perform data and projection transformations that are transparent to the user, providing the transformed data in a product-neutral format which could be read directly by Grassland's MapViewer component. However, it was not clear from the limited information available at the time whether use of this new product by Grassland in a local environment would result in any significant improvement in display speed. Therefore, the conventional approach was adopted of converting ADRG data into the standard Grassland internal representation using the L.A.S. utilities. The resulting files held the projection information used by the display driver (GRASS).

The performance on large image files has not been met; the resulting display speed for large raster files remains a question with no satisfying answer.

1.3.3 Questions on UNIX integration possibilities.

The resulting application demonstrated by the L.A.S. staff was running on a Windows platform using TCL/TK language which is very different from the ACTIF development setup (C++ and UIM/X). If this product was to be chosen, integration tests would need to be performed early in the development process to ensure satisfactory functioning of all components in the UNIX environment.

Tests have been conducted during the prototype development in order to integrate Grassland API into the ACTIF architecture. No problems were found using the Grassland API library (libmvx11.so) with C++, UIM/X and Pro*C (ORACLE).
We did not implement any OGDI or TCL/TK language in our prototype.

2.0 RASTER/VECTOR DISPLAY ALTERNATIVES

2.1 Goal

In order to test possibilities related to raster and vector map display, four setups have been successively experimented using the Grassland API:

FIGURE 2.1– Senior Analyst

a) Creation of a Grassland «Map Viewer» window within the existing «Senior Analyst» application;

b) Adding a vector type map layer into a «Map Viewer» window within the existing «Senior Analyst» application;

c) Adding a raster type map layer into a «Map Viewer» window within the existing «Senior Analyst» application;

d) Combining both vector and raster type in the same application.

2.2 Results Summary

This prototype revealed no problems implementing a mixed raster and vector solution. A unique API function handles all the display drivers letting us add different types of «layer» in a single «Map Viewer». This kind of function encapsulated in a C++ class put together with a UIM/X graphical user interface will be the path to follow in the ACTIF development.

Finally, this prototype provides the development team many ways to implement various ACTIF display modules; it also establishes the requirement for development purposes.

2.3 Approach Used

In order to display a map within the «Situation Map» application, Grassland requires some initialization. First, we must connect Grassland to our X11 window server. This step is handled by the Grassland API function 'MV_Init(*Display, screenNum*)' where *Display* represents the X Display and *screenNum* represents the X screen number. Fortunately UIM/X provides these values as UxDisplay and UxScreen.

Next, we need to create one or more «drawing» surfaces into our «Situation Map» window (X drawingArea widget). Grassland function 'MV_createView(*parentWnd, drawableWnd, x, y, args*)' creates an instance of a «Map Viewer» widget and links it to our UIM/X drawingArea. This function has been encapsulated in the WorkingSitMapView class where *parentWnd* represents the uiWorkingSitMapView class window identification and *drawableWnd* represents the window identification of a particular drawingArea widget.

Finally, the WorkingSitMapView class is instantiated when the user chooses the option Area of Operations Views... from the «Senior Analyst» menu Sitmap (Refer to Figure 1.1). The following example illustrates the previous concept (from here downward all code is written in C++):

```
WorkingSitMapView::SetX11Window()
{
char command[255];
int nbLayer;
int layer;
if(MV_Init(UxDisplay, UxScreen) != MV_SUCCESS)
  {
fprintf(stderr, "Couldn't initialize Map Viewer because: %s\n", MV_errorMessage());
exit(1);
  }
sprintf(command, "-width %d -height %d -background #cdcdc0c0b0b0",
     UxGetWidth( ((_UxCuiWorkingSitMapView *)gui)->drawingArea1),
     UxGetHeight( ((_UxCuiWorkingSitMapView *)gui)->drawingArea1));
viewer = (VIEW *) MV_createView(XtWindow(UxGetWidget(gui-
       >GenericSystemWindow)),
```

```
        XtWindow(UxGetWidget(((_UxCuiWorkingSitMapView  *)gui)-
        >drawingArea1)), 0, 0, command);
...
return(1);
}
```

A simple way to display vector information into a «Map Viewer» widget is to add a vector layer. This kind of operation is handled by the Grassland API function 'MV_addLayer (*view, args*)' where view represents one of our previous «Map Viewer» instances and args a command string. It is a common procedure within the Grassland API function to provide a command string (e.g., *args* in 'MV_addLayer' function) instead of many parameter options. In this case, *args* specifies the configuration options for a thematic map layer being added. Table 2.1 describes the possible values.

TABLE 2.1: VECTOR LAYER CONFIGURATION VALUES

Value	Meaning
description *title*	The *title* is a text string used to describe the map layer.
enable *Boolean*	Values are 0 (not enabled: map layer will not be displayed) and 1 (enabled: map layer will be displayed).
family *familyName*	Specifies the geographic type of the map layer. Valid *familyName* are: Matrix, Line, Area, Point and Text.
url *url*	Specifies the *url* pointer to the location where the coverage to be accessed is located. The URL is an ASCII string similar to the World Wide Web's Uniform Resource Locator. Valid *url* syntax is: gltp:[//<hostname>]/<name of driver>/<pathname>. – The <hostname> is used for remote driver access(e.g. //copernic.las.com). – Valid <name of driver> descriptors are: grass, vrf, dted, arcinfo, adrg and rpf (e.g., DIGEST geographic data type). – The <pathname> indicates the location of the Datastore.

Value	Meaning
description *title*	The *title* is a text string used to describe the map layer.
symbology *symbList*	Refer to Table 5.3 and section 5 of this document.
upscale *factor*	Specifies the highest threshold *factor* for displaying the map layer, where the highest *factor* is the smallest scale ratio.
downscale *factor*	Specifies the lowest threshold *factor* for displaying the map layer, where the lowest *factor* is the highest scale ratio.
opaque *Boolean*	Values are 0 (transparent mode) or 1 (opaque mode).

Once the command string is built, it is passed as a function parameter. The following sample code shows the approach used:

```
WorkingSitMapView::SetX11Window()
{
char command[255];
int nbLayer;
int layer;
...
//
   //Setup Grassland Demo
   //
/* Adding a Vector Layer called roads */
   sprintf(command, "-url
        gltp:/grass//export/home/auf1/grass/demo/spearfish/PERMANENT -
        coverage roads@PERMANENT(*) -family Line -symbology {{-type linesegment
        -color blue -width 3}}");
    if(MV_addLayer(viewer, command) )
  {
fprintf(stderr, "Couldn't add layer %s to Map Viewer because: %s\n", command,
        MV_errorMessage());
  }
/* Adding a Vector Layer called streams */
   sprintf(command, "-url
        gltp:/grass//export/home/auf1/grass/demo/spearfish/PERMANENT -
        coverage streams@PERMANENT(*) -family Line -symbology {{-type
        linesegment -color red -width 1}}");
    if(MV_addLayer(viewer, command) )
  {
```

```
fprintf(stderr, "Couldn't add layer %s to Map Viewer because: %s\n", command,
        MV_errorMessage());
  }
...
return(1);
}
```

Displaying raster information is similar to displaying vector information into a «Map Viewer» widget. Once again, the operation is handled by the Grassland API function 'MV_addLayer(*view, args*)' where *args* specifies the configuration options for a different thematic map layer being added. This time, the *familyName* value used is Matrix instead of Line and the *url* value descriptor used a raster driver. The next example shows the same API function handling a different kind of layer:

```
WorkingSitMapView::SetX11Window()
{
char command[255];
int nbLayer;
int layer;
...
//
    //Setup Grassland Demo
   //
// Adding an ADRG Raster turned into Grass format
      sprintf(command, "-url gltp:/grass//export/home/auf1/actif/PERMANENT -
         coverage xxca0103@PERMANENT(*) -family Matrix");
    if( MV_addLayer(viewer, command) )
  {
fprintf(stderr, "Couldn't add layer %s to Map Viewer because: %s\n", command,
        MV_errorMessage());
  }
...
return(1);
}
```

In order to manage different kinds of layers or more than one layer at a time, the «Map Viewer» widget maintains a drawing list including all layers. The previous function used, 'MV_addLayer', was simply adding a layer at the top of the drawing list. Some functions allow management of the content of a drawing list: modify ('MV_modifyLayer'), delete ('MV_deleteLayer'), draw

('MV_draw', 'MV_drawLayer'), reorder ('MV_reorderLayer') and change the current geographic display projection ('MV_setProjection'). These functions and many more will be discussed throughout this document. Finally, managing a drawing list of thematic layers allows us to implement both raster and vector maps on the same drawing surface without any difficulty.

2.4 Recommended Approach

At the time this document was written, it was clear that the UNIX version of the Grassland library (i.e., libmvx11.so) provided a mechanism for integration into the ACTIF architecture. UIM/X will handle the graphical user interface implementing a «drawing» surface for the «Map Viewer». The API functions will incorporate the required ACTIF coverage or layers into a drawing list. Encapsulated functions will provide a way to manage the drawing list. In that way, the current implementation will remain.

2.5 Implementation Rules (Annex B)

The reader will find the implementation rules and the necessary scripted routines at Annex B to perform the following functions:

a) Building Situation Map Windows; and

b) Creating Map Viewer instances.

3.0 IMPORT OF ADRG FILES FROM THE GEOMATICS CANADA CD-ROM

3.1 Goal

In order to let the «Senior Analyst» add ADRG coverage into a «Situation Map», the team managed to import ADRG files and tried to find the optimum resolution scheme for the best possible file format. The primary intention is to reduce the size and optimize the display of raster files. This section also addresses the capacity to read the files directly from CD-ROM and the size that such files required on disk.

3.2 Results Summary

Direct reading from a CD-ROM is possible but not recommended because the current CD-ROM technology does not offer a fast enough transfer capability. Moreover, ADRG files are huge and alternative file formats, such as CADRG (compressed ADRG), have to be used.

ADRG files have been converted into Grassland internal raster files (GRASS type). We used Grassland version 1.1 to import an ADRG file into a GRASS cell file because there was a driver problem with the Grassland version 1.2 beta (build 132).

3.3 Approach Used

At the time this document was written, Grassland products did not support file formats such as TIFF or JPEG. The current geographic data types available were: grass, vrf, dted, arcinfo, adrg and rpf. The optimization process is outlined below:

a) Load the ADRG file into Grassland application. The following command was executed using Grassland for UNIX version 1.1.

1) Open the librarian by selecting «Librarian» from the menu bar Object;
2) Select «Open Local...» from the menu bar File to create a local URL to a CD-ROM;
3) Select «ADRG(*.GEN)» from the List files Type and enter the path to the «*.gen» file (e.g./cdrom/cdrom0/adrg/xxca0101/xxca0101.gen);
4) Open a map viewer by selecting «Map Viewer» from the menu bar Object;
5) Drag and Drop the «*.img» from the Librarian to «Map Viewer» (e.g. xxca0103.img).

Statistics

An ADRG file xxca0103.img of ~213Mb was opened in a «Map Viewer» in approximately 12 minutes on a Sparc station 5 (6 minutes on a Sparc Ultra 140).

b. Create a new GRASS location header file. The following command shows how to create a new map set name based on the ADRG display of the «Map Viewer»:

1) Select «New Grass Location» from menu bar View located in the «Map Viewer» window;
2) Enter the «Target Directory» (e.g. $(HOME)/*grassLocation*);
3) Set the «Projection» parameter (e.g. +proj=utm +zone=13);
4) Set the North, South, East and West boundaries (e.g. -124.0016, 47.9997, -120.001 and 50.0001);
5) Enter the «Resolution» value (e.g. 0.00045);
6) Click «Create».

A new map set

$(HOME)/grassLocation/mapsetname/PERMENT was created containing the georeference file called «WIND».

c. Import the ADRG file to GRASS.

Before attempting to proceed with the last steps, you may need to restart Grassland and reconnect to the current URL(s) in the Librarian.

The import process needs to:

1) Select «Open Local...» from the menu bar File to create a local URL to the new created map set name in the Librarian;
2) Select «GRASS 4.1(WIND)» from the List of files Type and enter the path to the «WIND» file;
3) (e.g. $(HOME)/grassLocation/mapsetname/PERMENT/WIND);
4) Select the ADRG file (e.g. xxca0103.img) to convert;
5) Choose «Import to Grass» from the menu bar File;
6) Enter a new «Coverage Name» (e.g. xxca0103);
7) Identify the «Output Family» type (e.g. Raster);
8) Click OK.

At the end of the import process, update your map set by clicking the «Update» button in the Librarian that may now reference this new coverage like we saw in Table 2.2 (e.g., xxca0103@mapsetname(*)).

Statistics

An ADRG file xxca0103.img of ~213Mb was converted into a file of ~30Mb in Grassland internal raster format (GRASS) in approximately 54 minutes on a Sparc Ultra 140. It would require more than five (5) hours on a Sparc 5. To provide a useful point of reference, the geographic coverage of xxca0103.img was equal to three map sheets of 150km by 100km digitized at 1:250k.

3.4 Recommended Approach

At the time we wrote this document, the GRASS format was the approach best suited to increase display speed using the Grassland library.

The customer support told us that CADRG file format might be a better choice in our application domain. Tests were conducted using CADRG input files but ACTIF requirements were not met at the time. Finally, we recommended maintaining this approach until Grassland came up with new file formats (i.e., tiled / compressed format).

3.5 Implementation Rules (Annex B)

There are no rules to report for this function.

4.0 GIS TOOLBOX BUILDING

4.1 Goal

This section is dedicated to the integration of UIM/X graphical user interface windows with the Grassland API. The intent is to find ways to encapsulate the «Map Viewer» widget architecture in the main application. This section benefits from the knowledge acquired in section 2.

4.2 Results Summary

This activity illustrates the encapsulation of features such as scaling, zooming, querying distance and querying objects. The focus was on the display speed. The capability to achieve a fast display resides in the ability to «cache» in memory a large portion of a file shown in a window. For that reason, files have to be previously «cached» entirely once for every layer (coverage).

This ability was not yet supported for GRASS raster files during the prototype development.

4.3 Approach Used

In order to add some features to the «Map Viewer» window built in section 2, we created a «pop up» menu illustrated in Figure 3.1. This menu is accessed (i.e., shows up) by clicking the right mouse button anywhere on a «Situation Map».

The work of a «Senior Analyst» on a «Situation Map» brings out some common tasks. For the purpose of this prototype activity, we grouped them by mode: «Navigator», «Distance», «? Obj» and «Insert». To select a mode, a user accesses the toolbox and clicks the appropriate radio button. Table 4.1 describes each mode.

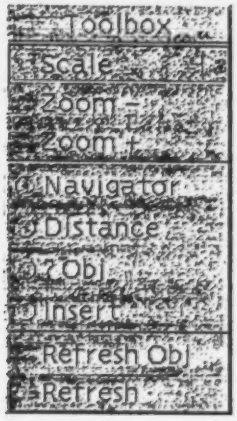

FIGURE 4.1– Demo toolbox

TABLE 4.1: TOOLBOX FUNCTION

Toolbox button	Description
Navigator	This mode enables a user to move his point of view over a «Situation Map». It activates scroll arrows located at the lower right end corner of the window (refer to Figure 4.7). Moreover, a user may select a new portion of the «Map Viewer» to zoom into.
Distance	This mode allows a user to get the distance between two or more points on a «Situation Map»
? Obj	This mode allows the user to interrogate and update any symbols (units) on the «Situation Map».
Insert	This mode allows the user to Insert symbols (units) on the «Situation Map».

Usually, the «Senior Analyst» gives input on the «Situation Map» via the mouse device. The mouse input on a «Situation Map» varies according to the current mode (i.e., which radio button is active). When the «Navigator» mode is turned on, each mouse click resets the viewer's region. However, in «Distance» mode, each click brings a distance measure shown in the bottom of the window (Refer to Figure 4.8). Finally, clicking on the «Situation Map» when either the «? Obj» mode or «Insert» mode is on specifies a unique pixel location (e.g. a unit).

The following table describes the features available under each toolbox mode.

TABLE 4.2: TOOLBOX MODE AVAILABLE FEATURES

	Mode	Description
Scroll arrows	Navigator	This feature allows the user to scroll the viewing region of the «Situation Map».
Interactive zooming	Navigator	This feature allows the user to manually redefine the viewing portion of the «Situation Map» (i.e., defining a rectangle).
Distance	Distance	This feature calculates the distance between two or more points on a «Situation Map».
? Obj	? Obj	This feature allows the user to interrogate and update any symbols (units) on the «Situation Map».
Insert	Insert	This feature allows the user to Insert new symbols (units) on the «Situation Map».
Scale	All	This feature changes the viewing scale of the «Map Viewer». Up to four (4) predefined values exist: 1:500k, 1:250k, 1:100k and 1:50k.
Zoom -	All	This feature allows the user to 'zoom out' using the «Map Viewer» by a factor 5:1.
Zoom +	All	This feature allows the user to 'zoom in' using the «Map Viewer» by a factor 1:5.
Refresh Obj	All	This feature allows the user to refresh the objects (units) layer manually.
Refresh	All	This feature allows the user to refresh the «Map Viewer» content manually.

4.3.1 Pan

Four predefined scales were integrated into the scaling function. Any scaling choice produces a redraw of the centre portion of the «Situation Map» at that scale. Moreover, the current scale is displayed at the bottom of the «Situation Map» shown in Figure 4.2.

FIGURE 4.2– Current scale display

```
void  WorkingSitMapView::vScaleMap(double  scale)
{
    if( MV_setScale(viewer, scale) )
  {
fprintf(stderr, «Couldn't set viewer to scale 1:%d because: %s\n», scale,
        MV_errorMessage() );
  }
   vRedraw();
}
```

Statistics

Depending on the map scale chosen, the performance of the scaling function varies as shown in Table 4.3.

TABLE 4.3: REFRESH TIME

Scale	Time to refresh			
	Sparc 5 (CADRG)	Sparc 5 (GRASS)	Ultra 140 (CADRG)	Ultra 140 (GRASS)
1:500000	~ 9 seconds	~ 3-4 seconds	~ 3 seconds	≤ 1 second
1:250000	~ 9 seconds	~ 3 seconds	~ 3 seconds	≤ 1 second
1:100000	~ 9 seconds	~ 2-3 seconds	~ 3 seconds	≤ 1 second
1:50000	~ 9 seconds	~ 2 seconds	~ 3 seconds	≤ 1 second

Note: For the purpose of this test, we compared 3 CADRG sheets (approximately 286kb per sheet) each cached in memory against the imported coverage obtained in section 3.

4.3.2 Zoom– / Zoom+

Each time a user chooses Zoom ±, the center portion of the «Situation Map» is reduced or enlarged by a factor 5 and the current scale is displayed.

```
void  WorkingSitMapView::vZoomInMap()
{
   if( MV_zoom(viewer, 0.2) )
  {
fprintf(stderr, «Couldn't zoom region for current viewer because: %s\n»,
        MV_errorMessage() );
  }
if( MV_getRegion(viewer, &n, &s, &e, &w, &res) )
  {
fprintf(stderr, «Couldn't get region for current viewer because: %s\n»,
        MV_errorMessage() );
return;
  }
vRedraw();
};
void  WorkingSitMapView::vZoomOutMap()
{
   if( MV_zoom(viewer, 5) )
  {
fprintf(stderr, «Couldn't zoom region for current viewer because: %s\n»,
        MV_errorMessage() );
  }
if( MV_getRegion(viewer, &n, &s, &e, &w, &res) )
  {
fprintf(stderr, «Couldn't get region for current viewer because: %s\n»,
        MV_errorMessage() );
return;
  }
vRedraw();
};
```

Statistics

The zooming function varies according to the current map scale as shown in Table 4.4.

TABLE 4.4: REFRESH TIME VERSUS MAP TYPES

	Sparc 5	Ultra 140
CADRG (RAM)	~ 9 seconds	~ 3 seconds
GRASS	~ 2-3 seconds	≤ 1 second

Note: For the purpose of this test, we compared CADRG sheets (approximately 286k per sheet) each cached in memory against the imported coverage obtained in section 3.

4.3.3 Scroll arrows

The user clicks on any arrow to move the «virtual window» over the «cached» layers. Section 4.3.9 explains the «cache» process. Depending on the selected arrow, the layers shift by a factor ±1/4 of the visible region of the «Map Viewer» (i.e., left, right or top down).

FIGURE 4.3– Navigator arrows

```
void  WorkingSitMapView::vScrollX(double  factor)
{
    if( MV_panX(viewer, factor))
  {
fprintf(stderr, «Couldn't pan the viewer vertically because: %s\n»,
        MV_errorMessage() );
  }

   vRedraw();
};

void  WorkingSitMapView::vScrollY(double  factor)
{
    if( MV_panY(viewer, factor))
  {
fprintf(stderr, «Couldn't pan the viewer horizontally because: %s\n»,
        MV_errorMessage() );
  }
vRedraw();
}.
```

Statistics

Depending on the current map scale, the scrolling function for a GRASS map varies from ~3-5 seconds on Sparc 5 and ≤ 1 second on ULTRA 140.

4.3.4 Interactive zooming

The user clicks on the «Situation Map» and drags a diagonal before releasing the mouse button (i.e., the 'button down' event anchors the initial point and the 'button up' event draws a rectangle from the anchor point to the 'button up' point). The new selected region (i.e., rectangle) of the «Situation Map» expands into the «Map Viewer».

```
void WorkingSitMapView::vInput(Widget widget, XtPointer call_data)
{
static  double dLat[32], dLong[32];
int   x, y;
char   buffer[64];
GC   gc;
XmDrawingAreaCallbackStruct *cbs = (XmDrawingAreaCallbackStruct *)call_data;
XEvent *event = cbs->event;
switch (mode)
{
        case NAV: if( event->xany.type == ButtonPress)
        {
                //Anchor initial point of the zoom area.
                if( MV_invTransform(viewer, event->xbutton.x, event->xbutton.y,
                &dLat[iPoints], &dLong[iPoints]))
                {
                        fprintf(stderr, «Couldn't get latlong for x=%d y=%d because:
                        %s\n», event->xbutton.x, event->xbutton.y,
                        MV_errorMessage() );
                        break;
                }
                iPoints++;
                }
        else if( event->xany.type == ButtonRelease )
                {
                int x2, y2;
                //Anchor final point of the zoom area.
                if( MV_invTransform(viewer, event->xbutton.x, event->xbutton.y,
                &dLat[iPoints], &dLong[iPoints]) )
                {
                        fprintf(stderr, «Couldn't get latlong for x=%d y=%d because:
                        %s\n», event->xbutton.x, event->xbutton.y,
                        MV_errorMessage() );
                        break;
                }
                 //Determine South value
                 if (dLong[0] < dLong[1])
                 {
                        n = dLong[1];
```

```
                s = dLong[0];
            }
            else
            {
                n = dLong[0];
                s = dLong[1];
            }
            //Search for West value
            if (dLat[0] < dLat[1])
            {
                e = dLat[1];
                w = dLat[0];
            }
        else
        {
            e = dLat[0];
            w = dLat[1];
        }
        //Get the upper left corner of the rectangle area.
        if( MV_transform(viewer, w, n, &x, &y) )
        {
            fprintf(stderr, "Couldn't get screen position for lat = %f long = %f
            because: %s\n", w, n, MV_errorMessage() );
            break;
        }
        //Get the upper left corner of the rectangle area.
        if( MV_transform(viewer, e, s, &x2, &y2) )
        {
            fprintf(stderr, «Couldn't get screen position for lat = %f long = %f
            because: %s\n», e, s, MV_errorMessage() );
            break;
        }
        //Create a Graphical context
        gc = XCreateGC(UxDisplay, cbs->window, 0, NULL);
        //Use XDrawRectangle primitive at x, y for width=x2-x and height=y2-y.
        XDrawRectangle(event->xany.display, cbs->window, gc, x, y, abs(x2-x),
    abs(y2-y) );
        //Set the new selected region.
        if( MV_setRegion(viewer, n, s, e, w, res) )
        {
            fprintf(stderr, «Couldn't set region for current viewer because: %s\n»,
            MV_errorMessage() );
            return;
}
        XFreeGC(UxDisplay, gc);
        gc = NULL;
        vRedraw();
        }
        break;
        default: break;
    }
}.
```

This sample code produced a visual feedback (i.e., a rectangle) before zooming. Section 4.3.5 describes how a similar visual feedback was handled for the distance feature.

Statistics

Depending on the current map scale the interactive zooming function for a GRASS map varies from ~3 seconds on Sparc 5 and ≤ 1 second on ULTRA 140.

4.3.5 Distance

The distance is calculated between the first and the last point location. The user defines those point locations by clicking on the «Map Viewer» (i.e., mouse input). Each click following the second one feeds two pieces of information back to the user. First, a segment is drawn over the «Map Viewer» between the last two points. Finally, a distance calculation is displayed at the bottom of the window (see figure 4.4). The prototype distance measure unit was the meter.

FIGURE 4.4– Distance Meter

```
void WorkingSitMapView::vInput(Widget widget, XtPointer call_data)
{
static  double dLat[32], dLong[32];
int   x, y;
char   buffer[64];
GC   gc;
       XmDrawingAreaCallbackStruct *cbs = (XmDrawingAreaCallbackStruct
       *)call_data;
       XEvent *event = cbs->event;
       switch (mode)
       {
       ...
              case DISTANCE: if( event->xany.type == ButtonPress)
              {
              //Anchor initial point
```

```
                if( MV_invTransform(viewer, event->xbutton.x, event->xbutton.y,
                &dLat[iPoints], &dLong[iPoints]) )
                {
                fprintf(stderr, «Couldn't get latlong for x=%d y=%d because: %s\n»,
                event->xbutton.x, event->xbutton.y, MV_errorMessage() );
                 break;
                }
                iPoints++;
        }
else if( event->xany.type == ButtonRelease )
        {
                double  dDistance;
                //Draw full line if more than one point
                if (iPoints-1)
                {
                //Reconvert previous x and y for drawing purpose
                if( MV_transform(viewer, dLat[iPoints-2], dLong[iPoints-2], &x, &y) )
                {
                        fprintf(stderr, «Couldn't get screen position for lat = %f long =
                        %f because: %s\n», dLat[iPoints-2], dLong[iPoints-2],
                        MV_errorMessage() );
                         break;
                }
                 //Create a Graphical context
                 gc = XCreateGC(UxDisplay, cbs->window, 0, NULL);
                 //Use XDrawLine primitive from previous x, y and last event x, y
                position
                 XDrawLine(event->xany.display, cbs->window, gc, x, y, event-
                >xbutton.x, event->xbutton.y);
                 //Calculate the overall distance
                 if( MV_calcDistance(viewer, iPoints, dLat, dLong, &dDistance) )
                 {
                        fprintf(stderr, «Couldn't calculate distance because: %s\n»,
                        MV_errorMessage() );
                 }
                 else
                 {
                        sprintf(buffer, «Distance = %.2f meter», dDistance);
                        vTrimRight(buffer, ' ');
                         UxPutLabelString(((_UxCuiWorkingSitMapView *)gui)-
                        >label1, buffer);
                 }
                 XFreeGC(UxDisplay, gc);
                gc = NULL;
                }
        }
        break;
        ...
        default: break;
  }
}.
```

The previous sample illustrated how Motif calls were encapsulated to produce visual feedback. The idea is to create a " graphical context " variable using the function 'GCvariable=XCreateGC'. Next, we used this variable as a parameter to another Motif function such as 'XDrawLine'. Finally, when no more is needed, the graphical context is destroyed with 'XFreeGC'.

4.3.6 ? Obj

The Query object process is detailed in Section 8.0 "Toolbox - Select and Query Functionality".

4.3.7 Insert

The insert process is detailed in Section 10.0 "Toolbox - Place Unit Symbol Functionality".

4.3.8 Refresh and Refresh Obj

Section 11.0 "Situation Map Refreshes" of this document details the «Map Viewer» refresh process.

4.3.9 Caching

The prototype encapsulates an API function called 'MV_loadcache' to load in memory a dedicated map layer. The implementation shown below illustrates the principle:

```
WorkingSitMapView::SetX11Window()
{
char command[255];
int nbLayer;
int layer;
        ...
        //Cache all layer in the Map Viewer
        if( MV_getNbLayer(viewer, &nbLayer) )
        {
                fprintf(stderr, «Couldn't get total layer number because: %s\n»,
               MV_errorMessage());
                nbLayer = 0;
        }
        for(layer=0;layer < nbLayer; layer++)
```

```
        if( MV_loadcache(viewer, layer) )
        {
                fprintf(stderr, «Couldn't cache layer %d because: %s\n», layer,
                MV_errorMessage() );
        }
        ...
return(1);
}
```

Note: The caching function handles only the vector type map. It is not yet implemented for GRASS raster type map. The customer support told us that it will be available by the end of the year (1998).

4.4 Recommended Approach

The functional aspect of the ACTIF system will clearly establish the graphical user interface need and feature requirements. For the purpose of this prototyping activity, we encapsulated the L.A.S. UNIX library in a subset of the object-oriented ACTIF architecture. We do not foresee any particular problem to re-use or adapt any feature developed during this activity. However, we recommend that the development team follow the Encapsulating L.A.S. API functions rule.

4.5 Implementation Rules (Annex B)

Use of the encapsulating L.A.S. API functional rules.

5.0 SYMBOLOGY ALTERNATIVE SOLUTIONS

5.1 Goal

This section is dedicated to study the available approaches to represent military symbols with the L.A.S. family of products. The following criteria must be addressed in the implemented solution:

a) Creation of symbol (multicolour);

b) Adding text to symbols;

c) Establishment of a library of symbols;

d) Manipulation of symbol primitives; and

e) Changing scale, visibility and appearance of symbols.

5.2 Result Summary

The «Dynamics» approach, available with Grassland, is a powerful solution and seems to offer everything that is required for the ACTIF system. There are some limitations in the current API package (version 1.2 beta) on the total number of dynamics (1024) and symbology descriptors (32). Nevertheless, no doubt is raised about the feasibility of the solution.

5.3 Approach Used

This section describes our implementation of the Grassland «Dynamics » procedures into a «Situation Map».

5.3.1 Symbology Requirements

The ACTIF system must be capable to handle multiple colour, text and basic symbols: marker, line or area. The following figure was taken from a NATO military symbols document [15]. It represents the rules, character length and positioning of each piece of information as shown on Figure 5.1 (fields B, C, D, E, F, G, H, J, K, L, M, N, P, Q, R, S, T, V & W) around a basic symbol (field A). See the documentation for complementary information.

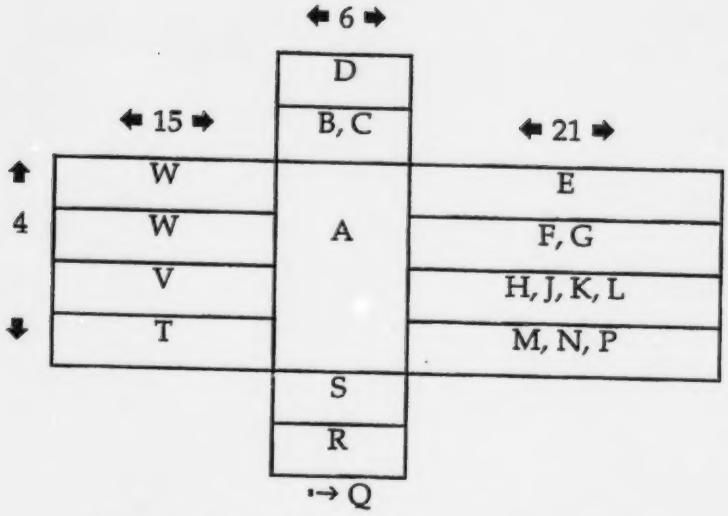

FIGURE 5.1– Military symbology fields

5.3.2 «Dynamic» Approach

A simple way to add information into a « Map viewer» widget is to create a «Dynamic» object. Grassland supports three (3) types of object through its procedures: point, line and area. Adding a «dynamic» is accomplished on a «cosmetic» layer by the Grassland API_function 'MV_addDynamic(*view, args*)' where *view* represents a «Map Viewer» instance and *args* a command string. In this case, *args* specifies the configuration options for any object being added. As seen in section 2, it is a common way to proceed. Table 5.1 describes the possible *args* values. An example of the encapsulation of such a function is given in section 7.

TABLE 5.1: POSSIBLE *args* VALUES

Value	Meaning
-id *dynamicId*	The *dynamicId* specifies a unique dynamic number (integer) greater than zero by which it is referenced.

-family *familyName*	Specifies the geographic type of the 'dynamic'. Valid *familyName* are: Line, Area and Point.
-coord *coordList*	A list of coordinate points according to the current projection. Each coordinate pair x and y is enclosed in braces. (e.g.{180 90}) The entire *coordList* is also enclosed in braces where each coordinate pair is separated by a space. (e.g. {{180 90} {123.45 678.9}})
-symbology *symbList*	Refer to Table 5.3.
-upscale *factor*	Specifies the highest threshold *factor* for displaying a dynamic object, where highest *factor* is the smallest scale ratio.
-downscale *factor*	Specifies the lowest threshold *factor* for displaying a dynamic object, where the lowest *factor* is the highest scale ratio.

Note: The current Grassland API package has a maximum of 1024 dynamics per «Map Viewer» widget.

4.3.3 Symbology List

In Grassland, each geographic type is represented graphically by one or more symbology descriptors enclosed in braces (i.e., *symbList*). Section 5.3.4 outlines the building process of a descriptor. A symbology list specifies the configuration options for any layer or object being added to a «Map Viewer». Table 5.2 describes the values while Table 5.4 describes the parameter values that apply to each symbol type.

TABLE 5.2: *symbList* VALUES FOR SYMBOL TYPE

Value	Meaning
-type *type*	Valid *type* (i.e., linesegment, icon, annotation, stipple or vertex) depends on the *familyName* option selected. Refer to Table 4.3
-color *color*	Specifies the foreground *color* to be used by the symbol.
-bitmap *@pathname*	Specifies a «.bmp» file. Depending on the type, the bitmap style became stipple, icon or hatch style.
-cgmfile *pathname* -iscgmscaled *Boolean* -cgmsize *size*	Specifies a «.cgm» file similar to the bitmap where *Boolean* identifies whether the CGM will be scaled according to the view scale. If not, the CGM will remain static. If the CGM is scaled, size represents the real size in the current unit projection or otherwise represents screen pixels.
-text *text*	Specifies *type* annotation text to be displayed.
-anchor *direction*	Valid *direction* values are: n, ne, e, se, s, sw, w, nw and center.
-xoffset *n*	Refer to Chapter 4: Displaying Map Layers [16]
-yoffset *n*	Refer to Chapter 4: Displaying Map Layers [16]
-angle *n*	Refer to Chapter 4: Displaying Map Layers [16]
-step *step*	Refer to Chapter 4: Displaying Map Layers [16]
-width *n*	Refer to Chapter 4: Displaying Map Layers [16]
-cap *type*	Refer to Chapter 4: Displaying Map Layers [16]
-join *style*	Refer to Chapter 4: Displaying Map Layers [16]
-font *font*	Specifies an X display server *font* to use when drawing text annotation.
-upscale *factor*	Specifies the highest threshold *factor* value for displaying the descriptor, where the highest *factor* value is the smallest

	scale ratio.
-downscale *factor*	Specifies the lowest threshold factor value for displaying the descriptor, where the lowest factor value is the highest scale ratio.

TABLE 5.3: *familyName* VALUES FOR SYMBOL TYPE

family *familyName*	type *type*				
	linesegment	icon	annotation	stipple	vertex
Area	X			X	X
Line	X	X	X		X
Point		X	X		X
Text			X		X
Matrix					

TABLE 5.4: PARAMETER VALUES FOR SYMBOL TYPE

Value	Type *type*				
	Linesegment	icon	annotation	stipple	vertex
color	X	X	X	X	X
bitmap	X	X		X	X
cgmfile	X	X		X	X
text			X		
anchor		X	X		X
xoffset	X	X	X		X
yoffset		X	X		X
angle		X	X		X

step	X	X	X		
width	X				
cap	X				
join	X				
font			X		
upscale	X	X	X	X	X
downscale	X	X	X	X	X

5.3.4 Symbology Descriptor

The following sub-sections will only cover the Point *familyName*. A symbology descriptor is one way to display a geographic type on a «Situation Map». In the prototype, we used the Computer Graphic Metafile (CGM) [17] format for the military representation. Therefore, each time a military element is inserted on a «Situation Map», a CGM file on disk is included within a descriptor clause. All descriptors come from a method called: WorkinSitMapView::chMakeSymbol(). This method uses the Oracle database to fill the parameter list and return a valid descriptor (e.g. «{*-type icon -color red -cgmfile somePath/unit.cgm -iscgmscaled 0 -cgmsize 50 -upscale 500000* }»). The following sample illustrates the concept of static descriptor. Section 5.3.7 discusses another type of descriptor built with the symbology procedure.

```
char * WorkingSitMapView::chMakeSymbol(char *symbol, char *type, char *color,
        int scaled, int size)
{
char element[128];
sprintf(element, "{-type %s -color %s -cgmfile /export/home/auf1/symbology/%s.cgm -
        iscgmscaled %d -cgmsize %d -upscale 500000}" \
, type, color, symbol, scaled, size);
return element;
}
```

Note: The current Grassland API package has a maximum of 32 descriptors grouped in a given symbology list.

5.3.5 Textual Information

Displaying text around a military symbol is another ACTIF requirement. To fulfil this requirement in the prototype, we had two possibilities: add text to a CGM file or create another symbology descriptor. In this prototype, we used the second approach because the textual information was taken from the Oracle database. Therefore, each «dynamic» object groups two types of descriptors: the representation (i.e., CGM files) and the annotation (i.e., textual information). The result is illustrated in Figure 5.4 where textual information (a date time group DTG) was added to military components.

5.3.6 Working with CGM

Computer Graphic Metafile is already a standard used by the US and Canadian Armies for the storage and transfer of picture description information (ISO/IEC 8632-1:1992). Grassland supports CGM files using a modified third party library from the Rutherford Appleton Laboratory (RAL). The current implementation of this library brings a certain limitation (see Note below). Nevertheless, CGM files will support all ACTIF needs.

All military symbols required by the ACTIF system must be created before their access. In this prototyping activity, we used our own CGM symbols and we made tests with the war fighting symbol library MIL-STD-2525A available on the Internet. For more convenience, we grouped all «*.cgm» in the same directory (i.e., library). A single CGM file stored on disk can be part of any number of descriptors at a time.

Note: In the current Grassland API package, CGM files are displayed using one colour at a time. Multiple colour schemes could be accomplished using multiple CGM for the same dynamic.

5.3.7 Managing symbols

The approach used during this prototyping activity let us add, modify, remove and refresh geographic components called layers or dynamics. This kind of approach is static because the symbology list parameter is determined at the time the components are added to the drawing list. Grassland offers a second way to proceed called symbology procedure. A symbology procedure is a function (such as: 'MV_setLayerSymbExec', 'MV_setDynamicSymbExec') called whenever a dynamic object or a layer is redrawn. These procedures are called once for each feature giving more flexibility in defining how individual features of the same type are displayed.

Note: Since this function will be called for every object contained in a coverage while the «Map Viewer» is being redrawn, drawing performance will be reduced.

Finally, Grassland API provides functions to group dynamic objects together, applying global symbology to the group. Refer to Chapter 10: Grouping Dynamic Objects [16].

5.3.8 Playing with descriptor thresholds

In the prototype, the descriptor threshold properties were used (i.e., upscale and downscale) to test the ACTIF architecture requirement. Therefore, we implemented features such as alternate symbology and scaleable details by using textual information.

First, we did experiment with alternate symbology at different scales. An annotation type descriptor was built to draw a coloured star «*» for scales over 1:500k (Figure 5.2) whereas at scale 1:250k another descriptor of type icon was displaying a CGM (Figure 5.3). The following sample code illustrates this feature.

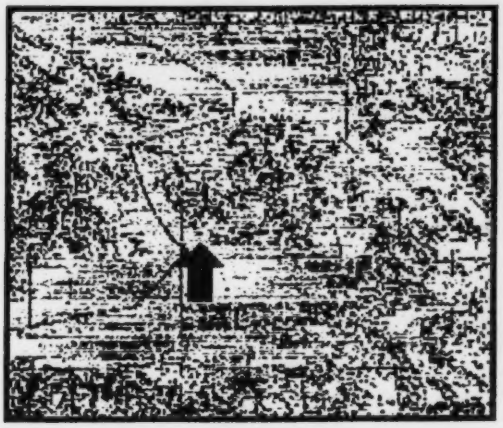

FIGURE 5.2 – Displaying alternate symbols at 1:500k

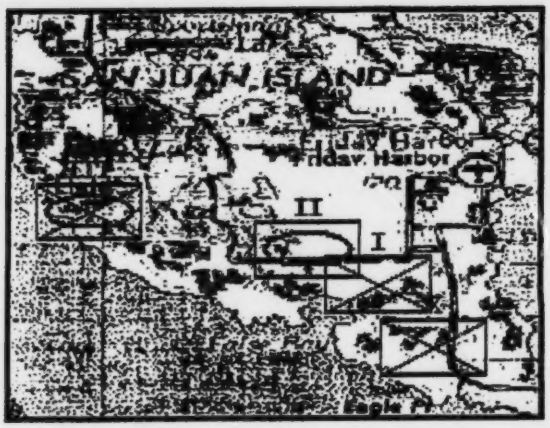

FIGURE 5.3 – Displaying symbols at 1:250k

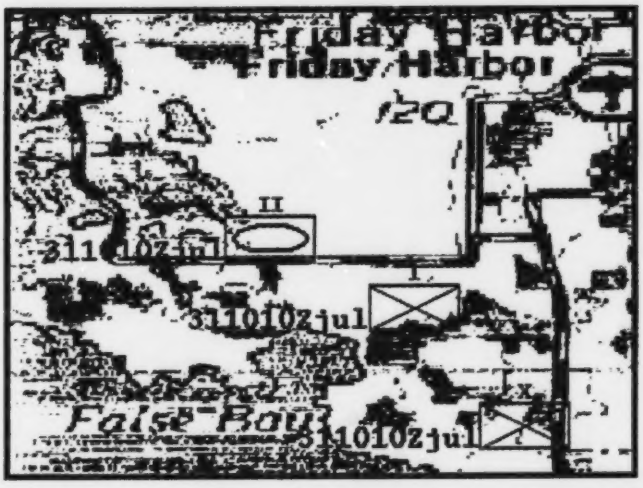

FIGURE 5.4 – Adding details to symbols at 1:100k

```
void  WorkingSitMapView::vCreateTactical()
{
char command[1024];
char symbol[512];
...
        //Create symbology identifier
        strcpy( symbol, chMakeSymbol(strDbNode.chCGM, «icon»,
        strDbNode.chColor, FALSE, 50) );
        /*
        * Construct a command string for a specific point
```

```
        symbol called a dynamic.
        */
        sprintf(command, «-id %d -family Point -coord {{%f %f}} -symbology {%s {-
        type annotation -color %s -text * -anchor center -font »«courier-Bold-12»« -
        upscale 3000000 -downscale 500000}}», strDbNode.iPosId, strDbNode.fPosX,
        strDbNode.fPosY, symbol, strDbNode.chColor);
...

}.
```

Next, we did add some textual details to a CGM symbol. Again, an annotation type descriptor was built to display the date-time group (DTG) of each dynamic. The DTG information was retrieved from an Oracle database. Figure 5.4 shows the addition of DTG information at scale under 1:250k.

```
void  WorkingSitMapView::vCreateTactical()
{
char command[1024];
char symbol[512];

...
        /*
        * Construct a command string for a specific point
        symbol called a dynamic.
        */
        sprintf(command, «-id %d -family Point -coord {{%f %f}} -symbology {%s {-
        type annotation -color black -text %s -anchor nw -xoffset -125 -font »«courier-
        Bold-16»« -upscale 250000}}», strDbNode.iPosId, strDbNode.fPosX,
        strDbNode.fPosY, symbol, strDbNode.chDTG);
...

}.
```

5.4 Recommended Approach

When this document was written, it was clear that Grassland combined with the Computer Graphic Metafile (CGM) would fully support all symbology needs for the ACTIF development. Therefore, the basics established through our approach would remain and would be implemented in an object-oriented architecture.

5.5 Implementation Rule

There are no rules to mention for this function.

6.0 SIMULATE THE CLASS ENCAPSULATION OF RDBMS

6.1 Goal

The objective of this activity is accessing a database by clicking on a military unit symbol on a working situation map. Some issues like response time and robustness were considered.

6.2 Result Summary

This method of accessing the Oracle database has been successfully implemented. L.A.S. does not provide any bridge to access an RDBMS; that is why this kind of approach was not considered. For architecture compliance, the class encapsulation has to be made directly in the «generic» part of C++ code.

6.3 Approach Used

We decided to re-use the knowledge of the ACTIF architecture development to implement a «three tier» approach. The graphical user interface tier is responsible for the display. The application tier is the main core of the ACTIF system. The database tier may be either a collection of raw data (a data store) or a database management system. Therefore, we did detach all database accesses from the GIS environment. A simple GUI is used to manipulate the symbols: uiObject.cpp class.

The Object class methods and attributes were grouped under the Object.pc file and access to Oracle: selection, insertion and update functions were encapsulated in the dbObject.pc file. We used PRO-C to pre-compile Object.pc and dbObject.pc into a super class called Object.

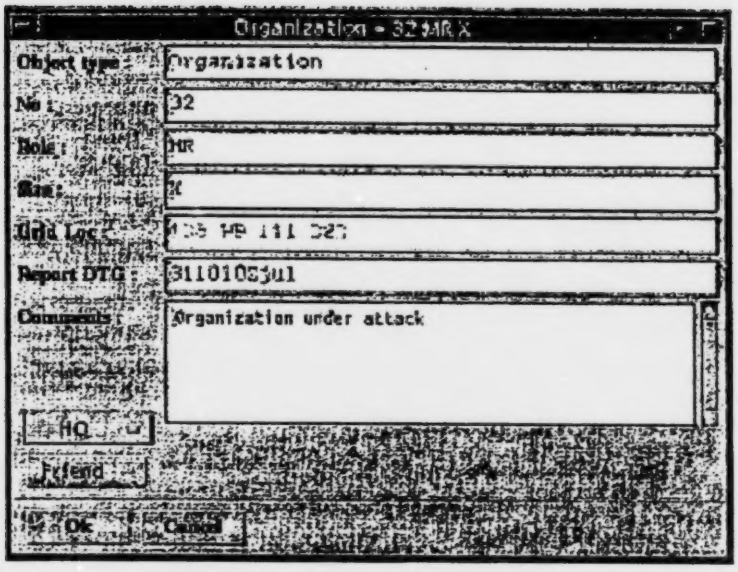

FIGURE 6.1- Symbology Dialog

TABLE 6.1: FIELD DESCRIPTION

Field	Description
Object type	25 characters describing an object (symbol).
No	25 characters identifying an object.
Role	25 characters giving an object role.
Size	25 characters describing the size of an object.
Grid Loc	This specific class will handle the grid location of an object (future development). Hard coded read-only for the prototype version.
Report DTG	25 characters indicating the date time group.
Comments	255 characters.

TABLE 6.2: BUTTON DESCRIPTION

Button	Description
[OK]	Submit the data to the database.
[Cancel]	Undo the modification.

TABLE 6.3: OPTION MENU DESCRIPTION

Option Menu	Description
HQ, Unit, Infantry, Armor and Recce	25 characters identifying a CGM filename representation.
Friend, Enemy and Neutral	5 characters giving an object color.

6.4 Recommended Approach

The result of our approach allows us to think that data duplication will not be necessary. The performance, time response and robustness were not an issue, as expected. The approach used, inspired by the ACTIF architecture, will remain and will be modified to support all types of Intelligence Objects.

6.5 Implementation Rules

The reader is advised to refer to "Coding Classes" of section 6 from Internal Design Standard [18].

7.0 INITIAL DISPLAY OF TACTICAL DATA

7.1 Goal

The main issue when initialising the session of a «Senior Analyst» is to rapidly process all the tactical database information from Oracle and to display the resulting symbols (units) on the «Situation Map». Refer to Figures 5.2 through 5.4.

7.2 Result Summary

The creation of a «dynamic» layer has been successfully implemented through the use of the API commands. Data is stored and managed in Oracle and converted (in «dynamics») only at the beginning of a given session. To achieve the implementation of such solutions, the software must provide good generating performance for the generation of tactical information. Grassland provides the kind of functionality and performance that are required.

7.3 Approach Used

The idea is to use our knowledge from section 5 and 6 to create «dynamics» reflecting the content of an Oracle database. We added dynamic objects to the «Map Viewer» for each unit (record) found in Oracle. A dynamic object uses an «id» to identify itself and one or more descriptors as its representation. The WorkingSitMapView::CreateTactical() method is responsible for the initial display of tactical data:

```
void  WorkingSitMapView::vCreateTactical()
{
char command[1024];
char symbol[512];
UxPutLabelString(((_UxCuiWorkingSitMapView *)gui)->label1, «Creating tactical
        data...»);
  if( !iOpenCursor() )
  {
```

```
        while( iFetch() )
        {
        //Remove trailing blanks
        vTrimRight(strDbNode.chSize, ' ');
        vTrimRight(strDbNode.chDTG, ' ');
        vTrimRight(strDbNode.chColor, ' ');
        vTrimRight(strDbNode.chCGM, ' ');
        //Create symbology identifier
        strcpy( symbol, chMakeSymbol(strDbNode.chCGM, «icon»,
        strDbNode.chColor, FALSE, 50) );
        /*
        * Construct the command string for a specific point
        symbol called a dynamic.
        */
        sprintf(command, «-id %d -family Point -coord {{%f %f}} -symbology {%s {-
        type annotation -color black -  text %s -anchor nw -xoffset -125 -font»«courier-
        Bold-16»« -upscale 250000} {-type annotation -color %s -text %s -anchor n -
        yoffset -25 -font»«courier-Bold-12»« -upscale 500000} {-type annotation -color
        %s -   text * -anchor center -font»«courier-Bold-12»«-upscale 3000000 -
        downscale 500000}}», strDbNode.iPosId, strDbNode.fPosX,
        strDbNode.fPosY, symbol, strDbNode.chDTG, strDbNode.chColor,
                strDbNode.chSize, strDbNode.chColor);
        /*
        * Insert a dynamic object at specified coordinate
        */
        if( MV_addDynamic(viewer, command) )
        {
                fprintf(stderr, «Couldn't add dynamic %d because: %s\n»,
        strDbNode.iPosId, MV_errorMessage() );
}
}
  }
 iCloseCursor();
  UxPutLabelString(((_UxCuiWorkingSitMapView *)gui)->label1, «Ready»);
 }.
```

Moreover, Grassland API provides functions to manage geographical object types as seen in section 5.3.7. We did not have time to implement symbology procedures within the time frame of this prototyping activity. Refresh of the tactical (i.e., «cosmetic») layer will be discussed in section 10.

Statistics

The total initial display time was between 3 and 10 seconds on a Sparc 5 to display a GRASS map sheet background (between 2-3 seconds on Ultra 140). Two Oracle tables were used: OBJECT and POSITION containing 300 records

each. The Oracle connection to SQL*net was established at the beginning of the «Senior Analyst» application. The vCreateTactical method adds ~2-3 seconds (Sparc 5) to the total initial display time (≤1 second on Ultra 140).

7.4 Recommended Approach

This prototyping activity only covered the geographic type Point. Nevertheless, we do not foresee any particular problem to go further with Line and Area types as they are similarly constructed. The ACTIF architecture will encapsulate the dynamic types and functions into its mapping symbology process. Further performance analysis would be required before using the symbology procedure approach (refer to section 5.3.7).

7.5 Implementation Rules (Annex B)

There are no rules to report for this function.

8.0 TOOLBOX – SELECT AND QUERY FUNCTIONALITY

8.1 Goal

The main goal of this activity is to rapidly access symbol information at a specific location, display the information in an independent window and update the information field if required.

8.2 Result Summary

The retrieval link to the dynamics created in section 7 "Initial Display of Tactical Data" has been successfully implemented. Because several alternative approaches to creating the initial display are possible, these 'select and query' functionality could have become issues. But the simple solution retained in section 7 made the implementation of 'select and query' easier. When selecting the «? Obj» toolbox mode, a user can select and query any dynamic object type within a «Map Viewer».

8.3 Approach Used

As seen in section 2, clicking on the «Situation Map» when the «? Obj» mode is active, refers to a unique location (e.g. a unit). This mode allows the user to locate any number of objects on the «Situation Map» by clicking the left mouse button. The Grassland function 'MV_getNearestDynamic' was implemented in the call back routine for mouse input on the «Map Viewer». This function allows the system to obtain the internal «dynamic id» number of the nearest dynamic object within a search criterion.

We used our knowledge of section 5 to retrieve the database information. Upon giving the «dynamic id» number, a selection is made in the Oracle database and the result is output (if any) into a separate window such as Figure 6.1. The following sample code illustrates how an (x and y) coordinate from the «Situation Map» (i.e., pixels) is converted into a «Map

Viewer» coordinate (i.e., latitude longitude). Finally, the nearest dynamic ID found in this projection system is used to retrieve information in Oracle.

```
void WorkingSitMapView::vInput(Widget widget, XtPointer call_data)
{
static  double dLat[32], dLong[32];
int   x, y;
char   buffer[64];
GC   gc;

XmDrawingAreaCallbackStruct *cbs = (XmDrawingAreaCallbackStruct *)call_data;
XEvent *event = cbs->event;

   switch (mode)
  {
...
       case QUERY: if( event->xany.type == ButtonRelease )
         {
              int     iDynamicId;
              int     iVertex;
              double   dTolerance = 0.5;

              if( MV_invTransform(viewer, event->xbutton.x, event->xbutton.y,
              &dLat[0], &dLong[0]) )
              {
                 fprintf(stderr, «Couldn't get latlong for x=%d y=%d because: %s\n»,
              event->xbutton.x, event->xbutton.y, MV_errorMessage() );
                 break;
              }
              if(MV_getNearestDynamic(viewer, dLat[0], dLong[0], dTolerance,
              &iDynamicId, &iVertex) )
              {
                 fprintf(stderr, «Couldn't get nearest dynamic because: %s\n»,
              MV_errorMessage() );
                 break;
              }
              if(iDynamicId > 0)
              {
                 vQueryObject(iDynamicId);
              }
                    else
                    {
                       sprintf(buffer, «No Object found within search radius %.0f
                    degree», dTolerance );
                       vTrimRight(buffer, ' ');
                        UxPutLabelString(((_UxCuiWorkingSitMapView *)gui)-
                    >label1, buffer);
                    }
               }
       break;
       ...
```

```
        default:  break;
    }
};
void  WorkingSitMapView::vQueryObject(int  Id)
{
MyObject *anObject;

    anObject = new MyObject(this, Id);
     anObject->vBuildWindw();
     anObject->gui->show();
};
```

Note: Section 11 gives more information on the coordinate management process and the military grid reference system.

When displayed, the information window is available for update. The «Senior Analyst» has two choices when the information window is updated. First, the user can confirm by clicking the «OK» button. The information is therefore sent to Oracle and the «cosmetic» layer is refreshed displaying the updated unit. Otherwise, the user can cancel the update process by clicking the «Cancel» button. The «cosmetic» layer is refreshed and the information window goes back to its original content.

8.4 Recommended Approach

The functional aspect of the future system will clearly establish the select and query functionality. When this document was written, we expected to use the «dynamic ID» number as the key to all our processing between Oracle, the GIS and the ACTIF system. Therefore, the 'MV_getNearestDynamic' approach seems to be the number-one option. Nevertheless, the final database and functional architecture may lead us to modify our approach slightly.

8.5 Implementation Rules (Annex B)

There are no rules to mention for this functionality.

9.0 TOOLBOX – PLACE UNIT SYMBOL FUNCTIONALITY

9.1 Goal

The work of a «Senior Analyst» involves introducing new objects into the ACTIF system. Those objects, such as event, organization or military symbols (e.g., units), modify the «Situation Map». This section benefits from the knowledge gained from section 7 for manipulating dynamic objects.

9.2 Result Summary

On one hand, the 'insert' function must manage the creation of dynamic type structures within the «Map Viewer» and the access to the RDBMS on the other. The coordinate list must be calculated from the feature position and then inserted into the RDBMS. The development team was able to prototype all the required functionality to insert and update features.

9.3 Approach Used

The 'insert' process takes place in two steps.

Location

As seen in section 2, clicking on the «Situation Map» when the «Insert» toolbox mode is active refers to a unique location (e.g. a unit). This mode allows the user to insert any number of objects on the «Situation Map» by clicking the left mouse button. Each time the user clicks on the «Map Viewer», a separate window «pops up» to complete the corresponding information.

Confirmation

The «Senior Analyst» has two choices when the information window is completed. First, the user can confirm by clicking the «OK» button. The information is therefore sent to Oracle and the «cosmetic» layer is refreshed, displaying a new inserted unit. Otherwise, the user can cancel the insertion

process by clicking the «Cancel» button. The «cosmetic» layer is refreshed and the 'insert' process is aborted. The following code illustrates the process:

```
void WorkingSitMapView::vInput(Widget widget, XtPointer call_data)
{
static  double dLat[32], dLong[32];
int   x, y;
char   buffer[64];
GC   gc;

XmDrawingAreaCallbackStruct *cbs = (XmDrawingAreaCallbackStruct *)call_data;
XEvent *event = cbs->event;

   switch (mode)
  {
...
       case INSERT :if( event->xany.type == ButtonRelease )
                {
                    if( MV_invTransform(viewer, event->xbutton.x, event-
       >xbutton.y, &dLat[0], &dLong[0]) )
                    {
                       fprintf(stderr, «Couldn't get latlong for x=%d y=%d because:
       %s\n», event->xbutton.x, event->xbutton.y, MV_errorMessage() );
                       break;
                    }
                    vInsertObject(dLat[0], dLong[0]);
                }
       break;
       default:  break;
  }
};

void WorkingSitMapView::vInsertObject(double posx, double posy)
{
MyObject *anObject;

       anObject = new MyObject(this, posx, posy);
      anObject->vBuildWindw();
       anObject->gui->show();
}.
```

9.4 Recommended Approach

The functional aspect of the future system will clearly establish this kind of functionality. The current prototype implemented a composite insertion in the «Map Viewer» and in Oracle at the same time. This

approach will remain in the ACTIF system. Nevertheless, the final database and functional architecture may lead us to slightly modify our approach.

9.5 Implementation Rules

There are no rules to mention for this functionality.

10.0 SITUATION MAP REFRESHES

10.1 Goal

We did identify two issues when working on a situation map. The first one was the initial display of the map (raster, vector and symbol data). It was addressed in Section 3 and 8. The second one is to refresh part or all the information contained in a «drawable» window of a situation map.

10.2 Result Summary

To achieve the implementation of such solutions, the software must provide good performance in the generation of tactical information. Grassland provides the type of functionality and performance that is required.

10.3 Approach Used

We divided the refresh process in two major sub-sections: window and symbols.

10.3.1 Window 'expose'

When working on an X11 window, some events may require repainting of part or all the window content. Those window events such as 'expose' and 'resize' are trapped by the UIM/X graphical user interface and a corresponding function is called.

When a window overlaps another window it hides certain parts of that window. The hidden part of that window will trigger an 'exposed' event to the system when it needs to be redrawn. The following sample illustrates the 'exposed' event response.

```
void  WorkingSitMapView::vRefresh()
{
    if( MV_repaint(viewer) )
```

```
  {
        fprintf(stderr, «Couldn't repaint the viewer because: %s\n»,
        MV_errorMessage() );
  }

    //Reset the number of points for calculated distance
   iPoints = 0;
};

void  WorkingSitMapView::vExpose()
{
    vRefresh();
}.
```

This very simple function encapsulates a Grassland API function called 'MV_repaint'. This function uses an X Motif «drawingArea» widget property called «backing store» to rapidly redraw the missing part of a covered window.

Statistics

This kind of refresh is accomplished under one second.

10.3.2 Window 'resize'

When a user decides to resize an X11 window, a 'resize' event is triggered and sent to the system asking to redraw its content. The following example illustrates the encapsulation of an API function called 'MV_configureView'

```
void  WorkingSitMapView::vResize()
{
char buffer[64];

   if( viewer != NULL)
  {
        sprintf(buffer, «-width %d -height %d», UxGetWidth(
        ((_UxCuiWorkingSitMapView  *)gui)->drawingArea1),UxGetHeight(
        ((_UxCuiWorkingSitMapView  *)gui)->drawingArea1));
       if( MV_configureView(viewer, buffer) )
    {
```

```
            fprintf(stderr, «Couldn't resize viewer to new values because: %s\n»,
            MV_errorMessage() );
        }

        vRefresh();
    }
}.
```

This function will resize the «Map Viewer» widget and re-compute all the projection parameters accordingly.

Statistics

This kind of refresh is as quick as any refresh occurring in the «Navigator» toolbox mode (refer to section 3.3.4 Interactive zooming).

10.3.3 Window 'redraw'

Finally, the layer content or the user point of view over the map may change in time. Therefore, certain manual or automatic updates would be required. This update was implemented using the API function 'MV_draw'.

```
void  WorkingSitMapView::vRedraw()
{
char  buffer[25];
double scale;

    if( MV_getScale(viewer, &scale) )
    {
        fprintf(stderr, «Couldn't get the current viewer scale because: %s\n»,
        MV_errorMessage() );
    }
    sprintf(buffer, «1:%.0f», scale);
     UxPutLabelString(((_UxCuiWorkingSitMapView *)gui)->label1, buffer);

    if( MV_draw(viewer,«») )
    {
        fprintf(stderr, «Couldn't redraw the viewer because: %s\n»,
        MV_errorMessage() );
    }

    //Reset the number of points for calculated distance
   iPoints = 0;
}.
```

This function will redraw the entire draw list (i.e., layers and dynamics) of the «Map Viewer» widget and re-compute all the projection parameters accordingly.

Statistics

This kind of refresh is as quick as any refresh occurring in the «Navigator» toolbox mode (refer to section 4.3.4 Interactive zooming).

10.3.4 Symbology Refresh

A symbology layer is built from two sources: an Oracle database (i.e., initial display of tactical data) and the «Senior Analyst» inputs (e.g., when the «Insert» mode is active). Other users connected to Oracle could modify the content of the «cosmetic» layer. Therefore, the ACTIF system will not only need to refresh the layers within the «Map Viewer» but all the symbology elements. This kind of refresh could be accomplished automatically by the system but, for simplicity, the prototype illustrated this concept manually from the toolbox function

«Refresh Obj». This function encapsulates two API functions such as 'MV_modifyDynamic' and 'MV_drawDynamicOnly'. These functions were encapsulated in the prototype method 'WorkingSitMapView::vRefreshTactical'. This method benefits from the knowledge of section 7 to refresh all dynamic objects at a time.

```
 void  WorkingSitMapView::vRefreshTactical()
{
char command[1024];
char symbol[512];

  if( !iOpenCursor() )
 {
      while( iFetch() )
```

```
{
    //Remove trailing blanks
    vTrimRight(strDbNode.chSize, ' ');
    vTrimRight(strDbNode.chDTG, ' ');
    vTrimRight(strDbNode.chColor, ' ');
    vTrimRight(strDbNode.chCGM, ' ');

    //Create symbology identifier
    strcpy( symbol, chMakeSymbol(strDbNode.chCGM, "icon",
strDbNode.chColor, TRUE, 1500) );

   /*
    * Construct the command string for a specific point
      symbol called a dynamic.
   */
    sprintf(command, «-id %d -family Point -coord {{%f %f}} -symbology {%s {-
type annotation -color black -text %s -anchor nw -xoffset -125 -font »«courier-
Bold-16»« -upscale 250000} {-type annotation -color %s -text %s -anchor n -
yoffset -25 -font»«courier-Bold-12»« -upscale 500000} {-type annotation -color
%s -text * -anchor center -font»«courier-Bold-12»« -upscale 3000000 -downscale
500000}}», strDbNode.iPosId, strDbNode.fPosX, strDbNode.fPosY, symbol,
strDbNode.chDTG, strDbNode.chColor, strDbNode.chSize,
strDbNode.chColor);

   /*
    * Update dynamic object at specified coordinate
   */
    if( MV_modifyDynamic(viewer, strDbNode.iPosId, command) )
   {
        fprintf(stderr, «Couldn't modify dynamic %d because: %s\n»,
strDbNode.iPosId, MV_errorMessage() );
    }
  }
}

 iCloseCursor();

 MV_drawDynamicOnly(viewer,«»);
}.
```

The previous sample shows how we read all units (records) stored in the Oracle database to reconstruct all symbology list descriptors before redrawing the «cosmetic» layer.

Statistics

This kind of refresh is accomplished under one second.

10.4 Recommended Approach

This prototyping activity showed that implementing all necessary call back routines within UIM/X gives the flexibility and the refresh performance needed by the ACTIF system. Therefore, this approach will remain.

At the time this document was written, we got good performance results implementing a global symbology refresh scheme. We did not foresee any problems implementing a different solution based on symbology procedures (refer to section 5.3.7). Nevertheless, the final database and functional architecture may lead us to slightly modify our approach.

10.5 Implementation Rules

There are no rules to mention for this functionality.

11.0 COORDINATE MANAGEMENT (MAP PROJECTION, ETC.)

11.1 Goal

The «Situation Map» will require transforming user-entered coordinate values from one map projection to another. The ACTIF architecture will also need to manage the Military Grid Reference System (MGRS) within its communication structure.

11.2 Results Summary

Grassland offers thirty-five (35) coordinate systems (projections) and twenty-three (23) ellipsoids. Therefore, we believe that it will be possible to encapsulate L.A.S. projection functions in a Grid Location class to support the Military Grid Reference System (MGRS) required within the ACTIF system. They also provide a script to transform (longitude, latitude) coordinates into MGRS coordinates.

11.3 Approach Used

The 'MV_createView' uses a projection descriptor to define its projection specifications. We did not build nor modify any particular projection specifications during this activity; we rather used the default value (i.e., -projection {+proj=longlat}).

Each time a «Senior Analyst» clicks on the «Map Viewer», the system must determine the input position of the mouse pointer. Within this prototyping activity, we had to transform «Map Viewer» pixel coordinates into (longitude, latitude) coordinates. Grassland provides two API functions to make those transformations possible such as: 'MV_transform' and 'MV_invTransform'. The following example shows how we manage this kind of constraint:

```
//Anchor initial point of the zoom area.
   if( MV_invTransform(viewer, event->xbutton.x, event->xbutton.y,
   &dLat[iPoints], &dLong[iPoints]) )
      {
   fprintf(stderr, «Couldn't get latlong for x=%d y=%d because: %s\n», event-
   >xbutton.x, event->xbutton.y, MV_errorMessage() );
break;
 }
```

11.4 Recommended Approach

We did not test different projection types in this prototyping activity. The approach used to transform coordinates will remain until new enhancement in the Grassland API package comes up.

11.5 Implementation Rules

There are no rules to mention for this functionality.

12.0 AREA SYMBOLS DISPLAY AND PLACEMENT

In section 9, we talked about inserting symbols on the «Situation Map». Part of the work of a «Senior Analyst» is also inserting symbols related to terrain features. Within Grassland, all symbols are called «dynamics». As we saw in section 5, the dynamic approach provides a uniform way of encapsulating symbology functions. Therefore, encapsulating these functions with MOTIF was not an issue, as expected. We did not have time to complete this section within the timeframe of the prototype development but we do not foresee any particular problem.

13.0 CONCLUSION

13.1 Context

The mission of the intelligence and surveillance combat function is to provide friendly forces with timely, accurate intelligence about hostile dispositions, capabilities and intentions, geographic conditions, targets and meteorology, while also denying friendly force information to an opponent. The provision of timely and accurate intelligence is an essential component in the Commander's decision-making process and the synchronization of battlefield combat power. It is a key element in the planning and conduct of all military operations. Thus, the rapid passage of information and intelligence to all levels of command is a vital and continuous activity.

Nowadays, effective computing support is needed to cope with the large flow of information generated by modern surveillance systems and troops in contact. Furthermore the uncertain character of the information such as inaccuracy, incompleteness and redundancy greatly complicates the task of timely intelligence analysis and strengthens the need for information technology. Also the priority of effort for the Canadian Forces has shifted from Central Europe, where a well-defined enemy and known mission fostered fixed organizations and standard operating procedures, to a less certain situation where neither the mission nor the enemy are known in advance. An Army intelligence system must be capable to quickly adapt to these new evolving situations and carry out its primary tasks in a flexible manner with a tailored intelligence capability appropriate to each mission element.

Any automated Command and Control Information System (C2IS), in order to be successful, must rely on an efficient geographical information system (GIS). Therefore, the purpose of this Technical Memorandum (TM) is

to present the results of prototyping activities aimed at exploring one technological alternative with the Genasys family of products.

13.2 Results

This study has demonstrated that the L.A.S. family of products would meet ACTIF needs subject to the limitations enumerated below. The following summarizes the strengths and weaknesses identified.

13.2.1 Strengths

INTEGRATION

Tests have been conducted during the prototype development in order to integrate the Grassland API (version 1.2 beta) into the ACTIF architecture. No problems were found using the Grassland API with C++, UIM/X and Pro*C under a UNIX environment.

RELIABILITY

We do not foresee any problems regarding the reliability of the product under a UNIX environment. The L.A.S. family of products will have the level of robustness that ACTIF requires.

SYMBOLOGY

The Grassland «dynamics» symbology approach combined with the Computer Graphic Metafile (CGM) will fully support all symbology needs for ACTIF development, including adherence to the military specifications.

13.2.2 Weaknesses

DISPLAY SPEED

The performance on large-image files (raster data) has not been demonstrated nor tested; the resulting display speed for large raster files remains to be evaluated.

SYMBOLOGY

The current Grassland API package (version 1.2 beta) has a maximum of 1024 dynamics per «Map Viewer» widget; that limitation may become a problem in the future.

SUPPORT

This prototyping activity showed that the L.A.S. family of products is not sufficiently documented and that the response time of their customer support was not effective. Programming schemes will have to be designed and followed by all developers to alleviate for this weakness.

13.3 Final Note

The Defence Research Establishment, Valcartier (DREV) has designed and produced automated tools to support the development of a modern tactical intelligence processing and analysis capability for the Canadian Land Force. Under the DREV "Advanced Concepts for Tactical Information Fusion" (ACTIF) project many studies were conducted to assist in the design of a future automated tactical intelligence system [5-10]. A new major R&D project D6479 LIEWA will contribute to define, validate and integrate concepts and requirements into Intelligence and EW technology demonstrators supporting the LF ISTAR architecture. This development is required due to an acknowledged Canadian Forces (CF) need for an intelligence and information technology investigator and evaluator to help reduce implementation risks of the capital acquisition L2634 ATISS project. It is hoped that the results of this study constitute a valid contribution that will enable the Intelligence Branch organization to produce a detailed design, to deliver and to implement a Tactical Intelligence System capable of embracing the Information Age.

14.0 REFERENCES

1– Gauvin, M., Lapointe, S. and Lizotte, M., "Tactical Information Fusion Prototype: Evaluation Through an Abbreviated Command Post Exercise", DREV R-9503, September 1995, UNCLASSIFIED

2– Gauvin, M. and Thibault, G., "Tactical Information Fusion Prototype: Participation in the ABCA Northern Lights Command Post Exercise", DREV R-9531, January 1996, UNCLASSIFIED

3– Thibault, G. and Beaumont, G., "Katimavik Project Phase I: Proposed TIF Prototype Modifications", DREV TM-9511, April 1996, UNCLASSIFIED

4– Thibault, G. and Beaumont, G., "Exercise Rite Complex 95, TIF Prototype Users' Comments", DREV TM-9511, August 1996, UNCLASSIFIED

5– "Development of a Strategic Plan for a Tactical Intelligence System", Final Report, DMR Consulting Group Inc., Contract W7701-6-2782, November 1997, UNCLASSIFIED

6– "Technological Infrastructure Architecture for a Tactical Intelligence System", Final Report, DMR Consulting Group Inc., Contract W7701-6-2782, November 1997, UNCLASSIFIED

7– "ACTIF Architecture, Database Design Description", Volume 1 and 2, DMR Consulting Group Inc., Contract W7701-6-4114, March 1997, UNCLASSIFIED

8– "ACTIF Architecture, System-Sub-System Design Description", DMR Consulting Group Inc., Contract W7701-6-2782, November 1997, UNCLASSIFIED

9– "ACTIF Architecture, Functional and Internal Standards", DMR Consulting Group Inc., Contract W7701-6-2782, November 1997, UNCLASSIFIED

10– "Concepts, Definitions and Requirements for a Future Tactical Intelligence System", DREV TM 9833, January 1999, CONFIDENTIAL

11– "GIS Study – Selection of technical Alternatives", ACTIF System Architecture Collection, DMR Consulting Group Inc., Contract W7701-5-1076, March 1996, UNCLASSIFIED

12– "GIS Study Part II – Selection of technical Alternatives", ACTIF System Architecture Collection, DMR Consulting Group Inc., Contract W7701-5-4114, December 1996, UNCLASSIFIED

13– Gibb, A., Tremblay, J., Turcotte, L., and Audet, F., "Selection of a Commercial GIS for Integration into a Command and Control Application - Lessons Learned from the ACTIF Intelligence Automation Project", DREV Technical Memorandum TM-9810, May 1998, UNCLASSIFIED

14– Graphics Reference Manual, Draft Version 1.0, Genasys II, December 1996, UNCLASSIFIED

15– B-GL-303-002/FP-Z01 Military Symbols, Volume 2, Supplement 1, Issued on Authority of the Chief of the Defense Staff, Canada, January 1989, UNCLASSIFIED

16– Visual Component (VIZ) documentation, Chapter 1-13 and Appendix A-D, prepared by Logiciels et Applications Scientifiques (L.A.S.) Inc., Montreal (Quebec), April 1997, UNCLASSIFIED

17– Henderson, L. R. and Mumford, A. M., "The CGM Handbook, Computer Graphics Metafile", Academic Press Limited, San Diego, California, 1993, UNCLASSIFIED

18– XSK-5-01835 Internal Design Standard, Standards for Software Design, prepared by DMR Consulting Group, Sainte-Foy, Quebec, October 1996. UNCLASSIFIED

19– Cockcroft, Adrian, "Sun Performance and Tuning", SunSoft Press, 1995. UNCLASSIFIED

ANNEX A – THE INTERNAL STRUCTURE

A.1 Directory

The main prototype program is located in the **actifPrototype2 / CommonFeatures / Application** and the GIS object classes are located in the **actifPrototype2/SituationMap/Gis directory.**

A.2 Library

The prototype library for the GIS module is located in:

actifPrototype2/SituationMap/Gis/lib/Gis.a

The CGM library is located in:

actifPrototype2/CommonFeatures/Application/exe/cgm/*.cgm

The Grassland shared objects are located in:

/opt/las/1.2/api/lib/

libadrg.so	libdted.so	libgrassext.so	libntdb.so	libtcl.a
libae.so	libdtusa.so	libgrassmod.so	libogdi.so	libvrf.so
libcgm.so	libecs_tcl.so	libmv.so	libremote.so	libzlib.so
libdtcanada.so	libgrass.so	libmvx11.so	librpf.so	

TABLE A.1: FILE DIRECTORY

Directory actifPrototype2/SituationMap/Gis/src/	Description
SpatialView.cpp	This class intended to reflect the C32_SpatialView of O_F32_VIEWS
MapView.cpp	This class is intended to reflect the C32_MapView of O_F32_VIEWS
SitMapView.cpp	This class is intended to reflect the C32_SitMapView of O_F32_VIEWS
uiWorkingSitMapView.cpp	This class represents the toolbox GUI designed for Demo1
WorkingSitMapView.pc	This Pro*C file encapsulates the class C32_WorkingSitMapView of

	O_F32_VIEWS. This is actually where all the Demo1 functionalities took place.
DbWorkingSitMapView.pc	This Pro*C file encapsulates the Oracle access for the initial display of the tactical information.
WorkingSitMapView.cpp	This class results from the pre-compiled WorkingSitMapView.pc and DbWorkingSitMapView.pc
uiObject.cpp	This class is used to display standard information of all symbols on the WorkingSitMapView
Object.pc	This Pro*C file encapsulates the class MyObject designed to Insert, Update and Query symbols on the WorkingSitMapView.
DbObject.pc	This Pro*C file encapsulates the Oracle access for Insert, Update and Select
Object.cpp	This class results from the pre-compiled Object.pc and DbObject.pc
uiOverlayList.cpp	This class was designed to control the visibility of the layers(GUI). *Not completed. Not yet implemented in any Demo.*
OverlayList.cpp	This class was designed to control the visibility of the layers(kernel). *Not completed. Not yet implemented in any Demo.*
ShownOverlay.cpp	This class was designed to control the visibility of the layers(layer object). *Not completed. Not yet implemented in any Demo.*

A.3 Makefile

To create the «Gis.a» library you need to create an appropriate Makefile in the **actifPrototype2/SituationMap/Gis/src/** directory:

```
include $(MAKE_DIR)/MakeActifVars

LIBNAME= Gis

PROCESSES=
```

```
CLASSES= SpatialView ShownOverlay uiOverlayList OverlayList MapView
      SitMapView
uiWorkingSitMapView WorkingSitMapView uiObject Object

SPEC_INCS= $(APPLICATION_INCS) $(INTEGRATION_INCS)
      $(UTILITIES_INCS) $(GUI_INCS)
 $(API_INCS)

include $(MAKE_DIR)/MakeActif
```

To compile the L.A.S. API program such as the prototype, be sure to have the following lines in your MakeActif and MakeActifVars:

actifPrototype2/actifUtils/MakeActif:

```
LAS_LIBS     = -lmvx11 -ltcl
LAS_LIBPATH   = $(API)/lib
...
LIBPATH        = $(X_LIBPATH) $(MOTIF_LIBPATH) -L$(UX_LIBPATH) -
      L$(LAS_LIBPATH) -L$(UCB_LIBPATH)   -R$(UCB_LIBPATH)
MOTIF_LIBS      = -lux $(LAS_LIBS) $(X_LIBS) -lw -ldl -lm -lsocket -lnsl_-lgen -lelf
      $(XOPEN_LIBS) -lc -lucb
...
$(EXE_DIR)/%: $(BIN_DIR)/%.o $(SYS_LIBS) $(EXE_DIR) \
         $(INTEVENTS_DIR)/lib/IntEvents.a \
          $(ORGANIZATION_DIR)/lib/Organization.a \
          $(INTEGRATION_DIR)/lib/Integration.a \
         $(MESSAGES_DIR)/lib/Messages.a \
        $(GUI_DIR)/lib/GUI.a \
          $(UTILITIES_DIR)/lib/Utilities.a \
          $(FILTER_DIR)/lib/Filters.a \
         $(GIS_DIR)/lib/Gis.a \
         $(LIB_DIR)/intManager.a
...
#
# Build an executable
#
$(PROCESSES): $(EXE_DIR)/$$@
$(EXE_DIR)/%: $(BIN_DIR)/%.o $(SYS_LIBS) $(EXE_DIR) \
            $(INTEVENTS_DIR)/lib/IntEvents.a \
             $(ORGANIZATION_DIR)/lib/Organization.a \
             $(INTEGRATION_DIR)/lib/Integration.a \
            $(MESSAGES_DIR)/lib/Messages.a \
            $(GIS_DIR)/lib/Gis.a \
            $(GUI_DIR)/lib/GUI.a \
             $(UTILITIES_DIR)/lib/Utilities.a \
             $(FILTER_DIR)/lib/Filters.a \
```

```
        $(LIB_DIR)/intManager.a
    @ echo Linking $(@F) ...
    @ $(RM) $@
    @ if $(PURIFY) $(CPP) $(CFLAGS) $(CPPFLAGS) -o $@ $(LIBPATH)
    $(BIN_DIR)/$*.o $(ACTIF_LIBS) $(ORACLE_LIBS) $(MOTIF_LIBS) -
    lxview -lolgx -lX11 -lm ; \
    then \
            touch $@ ; \
    else \
            $(RM) $@ ; \
    fi
```

actifPrototype2/actifUtils/MakeActifVars:

```
API_INCS= $(API)/include $(COMMON_INCS)

GIS_LIBS= $(GIS_DIR)/lib/Gis.a
GIS_INCS= $(GIS_DIR)/include $(COMMON_INCS)
...
ACTIF_LIBS=   $(APPLICATION_DIR)/lib/intManager.a $(COMMON_LIBS)
        $(MESSAGES_LIBS) $(ORGANIZATION_LIBS) $(INTEVENTS_LIBS)
        $(INTEGRATION_LIBS) $(FILTER_LIBS) $(UTILITIES_LIBS) $(GIS_LIBS)
        $(API_LIBS) $(GUI_LIBS) $(GR_LINK_LIBS)
...
#
#       ORACLE
#

PROCFLAGS= ireclen=255 oreclen=255 code=cpp select_error=no $(SQLCHECK)
        $(PROUSER) dbms=v7
        SYS_INCLUDE=\(/opt/SUNWspro/SC4.0.1/include/CC,/usr/include\)
        include=$(APPLICATION_DIR)/include  include=$(GUI_DIR)/include
        include=$(MESSAGES_DIR)/include  include=$(INTEVENTS_DIR)/include
        include=$(FILTER_DIR)/include  include=$(INTEGRATION_DIR)/include
        include=$(ORGANIZATION_DIR)/include
        include=$(UTILITIES_DIR)/include  include=$(MOTIFHOME)/include
        include=/opt/SUNWmotif/include  include=/opt/uimx2.9/include
        include=/usr/openwin/include  include=$(MOTIFHOME)/include/Xm
        include=$(MAIN_DIR)/include  include=$(ACTIFDB_DIR)
        include=$(GIS_DIR)/include  include=$(API)/include
```

A.4 Script SQL Grassland

Two scripts are available to test the prototype with Oracle:

TABLE A.2: ORACLE PROTOTYPE SCRIPTS

Directory actifPrototype2/CommonFeatures/Application/sql	Description
create.prototype.table.sql	Creates the required table in Oracle and inserts 6 symbols located around the center of the map for a rapid usage of the prototype.
populate.sql	Creates the required table in Oracle and inserts 300 symbols located in the bottom left of the map. It was designed to test the initial display of tactical data.

To run the script, type:
sqlplus **/@actifPrototype2/CommonFeatures/Application/sql/** script_name

A.5 Alias & Environment variables

To correctly install the L.A.S. product you must add some paths to your **$(HOME)/**.cshrc file:

```
setenv LAS          /opt/las/1.2
setenv GRASS        /export/home/auf1/grass
setenv GRASSLAND    $GRASS/sun_123
setenv GISBASE      $GRASS/Grass4.1
setenv GISRC        $GRASS/ .glrc
setenv API          $LAS/api
setenv LD_LIBRARY_PATH    $GRASSLAND/bin:$API/lib
setenv PATH .: $LAS
```

To help you launch the prototype executable, you could add an alias into your **$(HOME)/**.actifrc file:

setenv GIS_DIR *$ACTIFHOME/SituationMap/Gis*
alias intManager *ActifPrototype2/CommonFeatures/Application/exe/intManager*

To correctly enable the .cshrc or .actifrc type source **$(HOME)/**.cshrc

A.8 Compile

To compile the prototype type: **actifPrototype2/**Make

A.9 Run

The prototype demonstrates some GIS functions available via a toolbox integrated to the «Situation Map» view of a «Senior Analyst». Type intManager to run it.

ANNEX B – IMPLEMENTATION RULES

B.1 Designing GUI – Building Situation Map Windows

B.1.1 What?

In L.A.S., X11 windows are used for graphics display. Multiple GIS windows can be created having a unique identifier within the window server. Grassland provides a way to initialize the GIS window with the API package.

B.1.2 When?

In ACTIF, a GIS window should be created each time a user needs a GIS display representation. Therefore, the user will be able to move, raise, «iconize» and restore this window. The window is removed when the user no longer displays GIS information.

B.1.3 How?

a) Create a new window using the 'Generic System Window' widget from the ACTIF palette.

b) Go into the property editor and change the following properties.

TABLE B.1: PREPERTY EDITOR VALUES

Section	Property	Value
Specific	Background	UxGetBackground(«SYST», «background»);
Specific	DialogTitle	New Title of the window.
Declaration	Name	'ui' + < Domain class name >

c) Add a «drawingArea» swidget (UIM/X implements MOTIF widget as swidget) into the new window created above.

d) Go in this form property editor and change the following properties.

TABLE B1.2: PROPERTY EDITOR PROPERTIES

Section	Property	Value	Source
Core	Background	«black»	-
Constraint	BottomAttachment	attach_form	Private
Constraint	LeftAttachment	Attach_form	Private
Constraint	RightAttachment	Attach_form	Private
Constraint	TopAttachment	Attach_form	Private

e) Under the domain member section class initializing the window, write

```
/*
 * Initialization of the Grassland API package to UIMX display server.
*/
if(MV_Init(UxDisplay, UxScreen) != MV_SUCCESS)
 {
      fprintf(stderr, «Couldn't initialize Map Viewer because: %s\n»,
      MV_errorMessage());
      exit(1);
 }
```

B.2 Creating Map Viewer Instances

B.2.1 What?

In L.A.S., «Map Viewers» are used for displaying coverage and dynamics. Multiple «Map Viewer» instances can be created within a single GIS window having a unique identifier. Grassland API functions use a «Map Viewer» instance as function parameter.

B2.2 When?

In ACTIF, a «Map Viewer» instance should be created each time a user needs to display a map, a coverage or military symbol. Therefore, the user will be able to manage this kind of information using API functions on a

specific «Map Viewer». The «Map Viewer» instance is removed when the user no longer needs it.

B2.3 How?

Under the domain member section class initializing the window, write

```
/*
 * Create a Map Viewer instance.
 */
char   command[255];
VIEW *viewer;

sprintf(command, «-width %d -height %d -background %s», width, height,
        colorName);

        viewer = (VIEW *) MV_createView( XtWindow( UxGetWidget( gui-
        >GenericSystemWindow) ), XtWindow( UxGetWidget( ( (_UxCuiClassName*
        )gui )->drawingAreaName ) ), x, y, command );
```

B.3 Integrating GIS Components into ACTIF

Encapsulating L.A.S. API functions

B.3.1 What?

A GIS method is a possible action to accomplish over certain GIS classes. Defining a method is described in section 6.3 of the Internal Design Standard [18].

B.3.2 When?

Each Grassland API function needed by the application should be encapsulated in an ACTIF GIS class. In L.A.S., most of the API functions are built to return a Boolean value such as: TRUE or FALSE.

B.3.3 How?

The programmer is able to avoid function failure by testing the return value of the function. Refer to section 12.4 Displaying System Messages of the Internal Design Standard [18] to handle the error message.

```
/*
 * Grassland API function returning int type.
 */
if(MV_someFunction(parameters...) != MV_SUCCESS)
  {
        fprintf(stderr, «Some Error message: %s\n», MV_errorMessage());
        exit(1);
  }
```

ANNEX C – PERFORMANCE & STATISTICS

C.1 Testing Environment

Some analyses were carried out on the «Senior Analyst» prototype through the use of system activity reports. The UNIX program 'sar' [19] was used to collect processor, disk and memory activity each 5 seconds. Refer to the UNIX 'man' page for detailed information on this command.

These tests were made under the following environment:

Sparc station 5

Microsparc 85 MHz

Video card Turbo GX

64 Mb RAM

Internal disk (SD3) 1.05 Gb

External disk (SD2) 2.1 Gb

128 Mb swap space located on SD3

Solaris 2.5

Application and User account stored on SD2

Each activity reading results from a graphical user interface on a raster file of 30Mb and a «cosmetic» layer. Actions have been conducted following this schedule:

TABLE C.1: DEMO 1 EXPERIMENTATION SCHEDULE

Time	Action
11:09:45	The Demo is launched from the **actifPrototype2/CommonFeatures/Application/exe/intManager**.
11:10:00	Option «Area of Operations Views» is selected from menu bar Sitmap. The Working sit map view is displayed.
11:10:20	The toolbox function scale 1:500k is selected.
11:10:30	The toolbox function scale 1:250k is selected.

11:10:40	The toolbox function scale 1:100k is selected.
11:10:50	The toolbox function scale 1:50k is selected.
11:11:10	The toolbox function scale 1:500k is selected.
11:11:20	The toolbox function scale 1:100k is selected.
11:11:30	The toolbox function scale 1:250k is selected.
11:11:40	The toolbox function scale 1:50k is selected.
11:12:00	The Zoom In and Out function are called by clicking either Zoom + or Zoom -
11:13:30	The viewer position on the map changes according to the selected arrow direction. Radio button mode is on Navigator.
11:15:00	Some distance calculations are made while clicking segments on the map. Radio button mode is on Distance.
11:16:30	The function 'Zoom interactive' is called by defining a new viewer portion of the map. Radio button mode is on 'Navigator'.
11:17:00	The toolbox function 'query object' is called when selecting a symbol on the map. Radio button mode is on '? Obj'.
11:17:30	An Oracle update is perform by clicking the Ok button within the symbology dialog (refer to figure 5.1)
11:18:00	A new symbol is inserted into the «Map Viewer» by clicking its position. Radio button mode is on 'Insert'.
11:18:20	A new symbol is inserted into an Oracle database by clicking the Ok button.
11:18:40	An Oracle update is perform by clicking the Ok button within the symbology dialog (refer to figure 5.1)
11:19:00	The toolbox function 'refresh symbology' is called by clicking 'Refresh Obj'.
11:19:10	The toolbox function 'refresh symbology' is called by clicking 'Refresh Obj'.
11:19:30	The toolbox function 'refresh map content' is called by clicking 'Refresh'.
11:19:40	The Working Sitmap window size is maximized using ALT+F10
11:20:09	The Working Sitmap window size is restored using ALT+F5
11:20:30	The Working Sitmap window is resized by grabbing any border.
11:21:00	The demo is terminated.

C.2 Terminology Definitions

Runnable Queue: 'runocc'

The 'runnable' queue is the primary measure that can be used to decide whether a system needs more CPU power. If a process is ready to run but there is no free CPU to run on, then the process waits in this queue. On a multiprocessor machine, the queue empties more quickly than on a unique processor. A CPU time slice is about 50 to 100 ms, and a service time is up to four time-slices is usually perceived to be acceptable performance. 'runocc' is the proportion of the time that there is something in the run queue [19].

Swap Space: 'freeswap'

'freeswap' shows the free swap in 512-byte blocks. 'freeswap' is equal to available swap + reserved swap [19].

Free Memory: 'freemem'

'freemem' reports memory in pages of RAM that are immediately ready to be used whenever a process starts up or needs more memory [19].

Minor Faults: 'vflt'

A minor fault is caused by an address space or hardware address translation fault [19].

Other Fault Types: 'pflt'

Protection fault are caused by illegal access of the kind that produce "segmentation violation – core dumped" messages [19].

Pages Paged In: 'pgin', 'ppgin'

The number of page faults and the number of pages paged in by swap space or file system reads. Since the file system block size is 8 Kbytes, there are often two pages or 8 Kbytes, paged in per page fault [19].

Pages Paged Out: 'pgout', 'ppgout'

The number of page-outs and the number of pages paged out to the swap space or the file system. Because of the clustering that occurs on swap space writes, there may be very many pages written per page-out [19].

Pages Freed: 'pgfree'

Pages freed is the rate at which memory is being put onto the free list by the page scanner deamon. The rate is in pages freed per second [19].

The Page Deamon Scanning Rate: 'pgscan'

These commands report the number of pages scanned by the page deamon as it looks for pages to steal from processes that are not using them often. This is the key memory shortage indicator if it stays above 200 pages per second for long periods [19].

The Activity data reported: 'avserv', 'avwait', 'avque'

'AVSERV' Average time, in milliseconds, for a transfer request to be completed by the device (for disks, this include seek, rotational latency and data transfer times).

'AVWAIT' Average time, in milliseconds, that transfer requests wait idly in the queue (measured only when the queue is occupied).

'AVQUE' The sum of the average wait time plus the average service time.

C.3 Test Results

The following analyses are followed by summaries and graphic reports on each activity.

C.3.1 Demo 11:09:45

This «Situation Map» demo was developed around the «Senior Analyst» graphical user interface prototype. The biggest memory need was at the initialization of the «Senior Analyst» and the «Map Viewer»: around 2.8 Mb (Figure C.1). The application had an average memory need for 2.6Mb throughout. The maximum swap space used was 6.2 Mb (for a refresh) and the swapping area had an average size of 1 Mb (Figure C.2). The processor activity tells us that 29% of the time there was something in the run queue. The results reveal much about this prototype approach. Results show that each action (i.e. zoom ±, pan, etc.) needs to free space (ppgout/s, pgout/s) and load (ppgin/s, pgin/s) the required data in memory. This approach may need some processing time when the system searches for a memory page (pgscan/s), resolves faults (vflt/s, pflt/s) and allocates pages in the swapping area (blks/s(internal)).

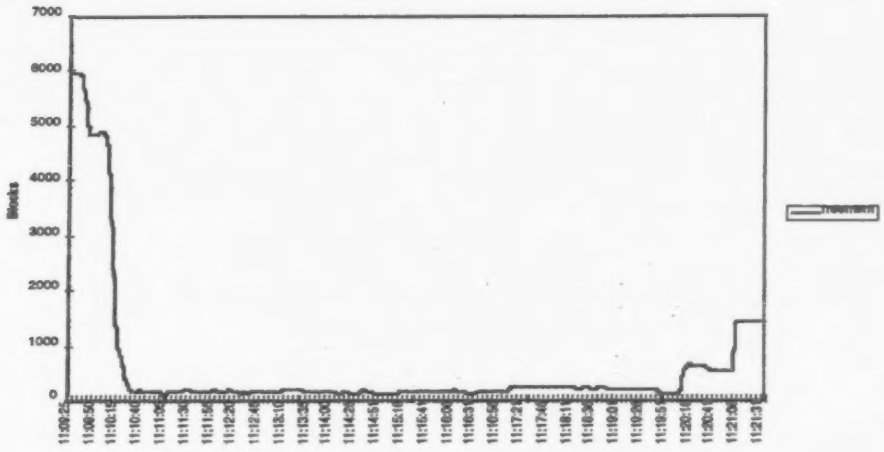

FIGURE C.1– Demo Free Memory

'freemem' reports memory in pages of RAM that are immediately ready to be used whenever a process starts up or needs more memory. The biggest memory need was at the initialization of the «Senior Analyst» and the «Map Viewer»: around 2.8 Mb. The application had an average memory need for 2.6Mb throughout.

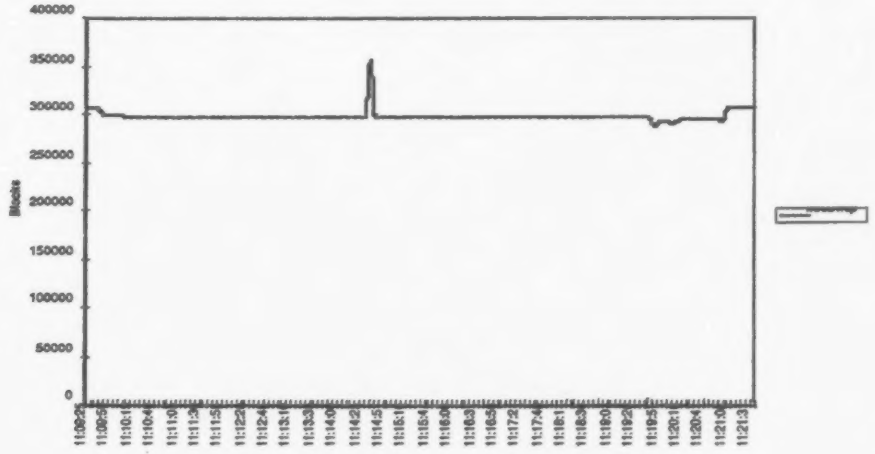

FIGURE C.2– Demo Free Swap

'freeswap' shows the free swap in 512-byte blocks. 'freeswap' is equal to available swap + reserved swap. The maximum swap space used was 6.2 Mb (for a refresh) and the swapping area had an average size of 1 Mb.

C.4 Result Summary for Part I

C.4.1 Processor activity average

TABLE C.2: PROCESSOR ACTIVITY

%usr	%sys	%wio	%idle	%runocc
19.9	5.6	3.5	71	29.3

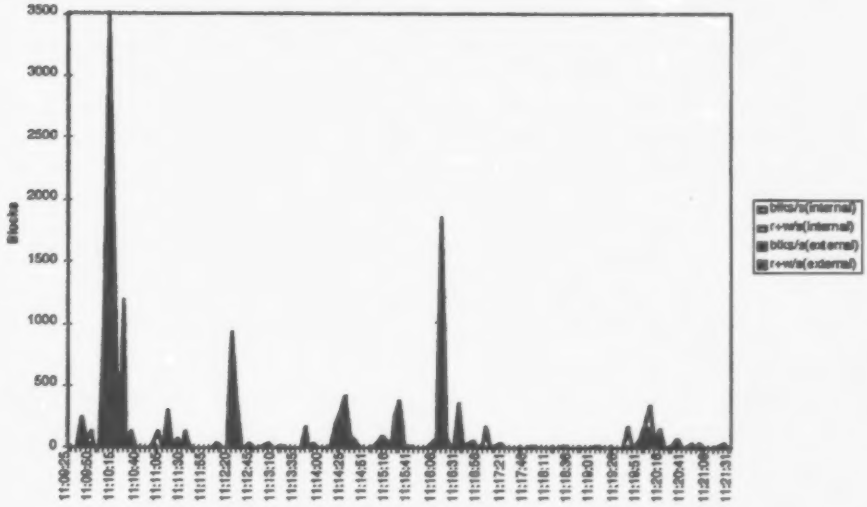

FIGURE C.3 – Disk I/O

C.4.2 Disk Activity Average

TABLE C.3: DISK ACTIVITY

device	blks/s	r+w/s	avserv ms	avwait ms	avque ms
SD2 (external disk)	88.4	3.4	16.6	0	0.1
SD3 (internal disk)	20.2	0.6	15.6	0	0

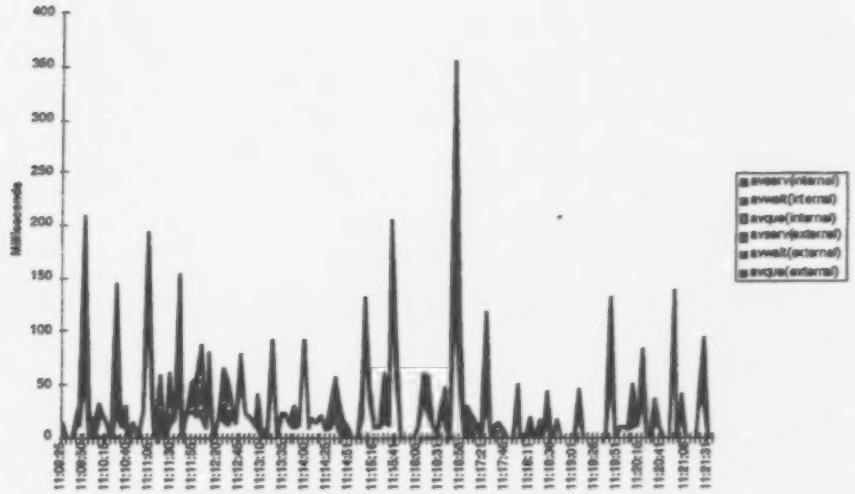

FIGURE C.4– Disk Activity

C.4.3 Memory Activity Average

TABLE C.4: MEMORY ACTIVITY

pgout/s	pgin/s	ppgout/s	ppgin/s	vflt/s	pflt/s	pgscan/s	pgfree/s	freemem[1]	freeswap[2]
0.1	3.5	1.6	11.9	17.6	1.2	18.4	5.4	608.4	297943.5

[1] Reports memory in pages of RAM

[2] Reports free swap in 512-byte blocks

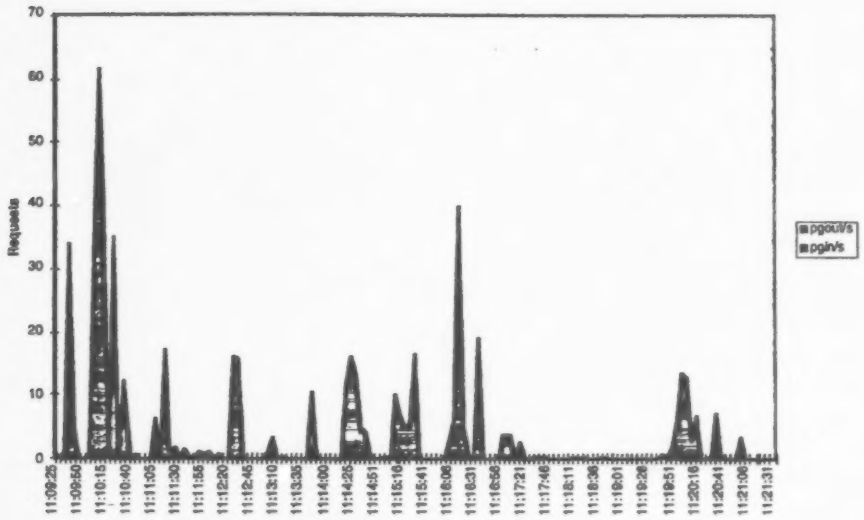

FIGURE C.5– Memory Request Activity

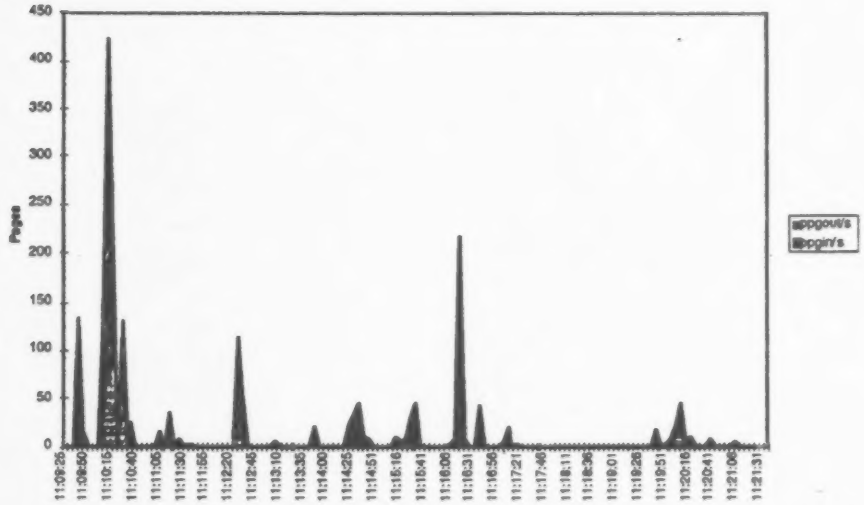

FIGURE C.6– Paging Activity

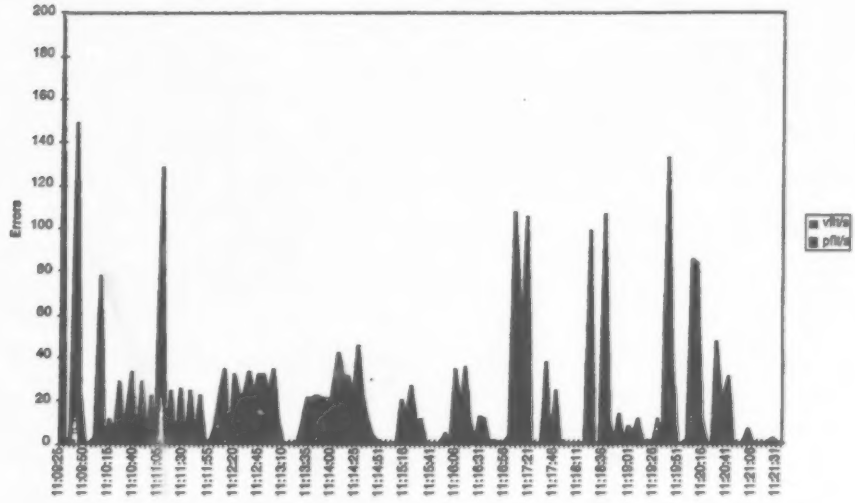

FIGURE C.7– Page Faults

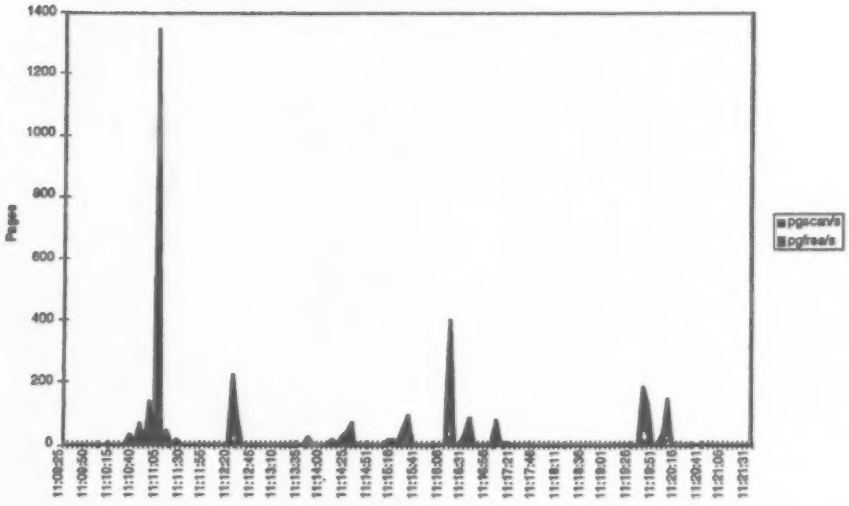

FIGURE C.7– Page Stealing Activity

C.5 Result Summary for Part II

C.5.1 Import of ADRG files from the Geomatics Canada CD-ROM

An ADRG file xxca0103.img of ~213Mb was opened in a «Map Viewer» in approximately 12 minutes on a Sparc station 5 (6 minutes on a Sparc Ultra 140).

An ADRG file xxca0103.img of ~213Mb was converted into a file of ~30Mb in Grassland internal raster format (GRASS) in approximately 54 minutes on a Sparc Ultra 140. It would require more than five (5) hours on a Sparc 5.

C.5.2 GIS Toolbox

Depending on the map scale, the performance of the scaling function varies as shown in TABLE C.5.

TABLE C.5: MEMORY ACTIVITY

Scale	Time to refresh			
	Sparc 5 (CADRG)	Sparc 5 (GRASS)	Ultra 140 (CADRG)	Ultra 140 (GRASS)
1:500000	~ 9 seconds	~ 3-4 seconds	~ 3 seconds	≤ 1 second
1:250000	~ 9 seconds	~ 3 seconds	~ 3 seconds	≤ 1 second
1:100000	~ 9 seconds	~ 2-3 seconds	~ 3 seconds	≤ 1 second
1:50000	~ 9 seconds	~ 2 seconds	~ 3 seconds	≤ 1 second

Depending on the current map scale the zooming function varies as shown in TABLE C.6.

TABLE C.5: MEMORY ACTIVITY

Raster File Format	Sparc 5	Ultra 140
CADRG (RAM)	~ 9 seconds	~ 3 seconds
GRASS	~ 2-3 seconds	≤ 1 second

Depending on the current map scale the scrolling function for a GRASS map varies from ~ 3-5 seconds on a Sparc 5 (≤ 1 second on ULTRA 140).

Depending on the current map scale the interactive zooming function for a GRASS map varies from ~ 2-3 seconds on a Sparc 5 (≤ 1 second on ULTRA 140).

C.5.3 Initial display of tactical data

The total initial display time was between 3-10 seconds (Sparc 5) to display a GRASS map sheet background (between 2-3 seconds on Ultra 140). Two Oracle tables were used: OBJECT and POSITION containing 300 records each. The Oracle connection to SQL*net was established at the beginning of the «Senior Analyst» application. The vCreateTactical method adds ~2-3 seconds (Sparc 5) to the total initial display time (≤ 1 second on Ultra 140).

C.5.4 Refreshes

Most of the refresh schemes were accomplished under one second.